Escola Politècnica Superior
d'Enginyeria de Vilanova i la Geltrú
UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE FI DE CARRERA

TÍTOL: Lectums: red social para amantes de la lectura

AUTOR: Alexandre Hernández Trueba

TITULACIÓ: Ingeniería Técnica en Informática de Gestión

DIRECTOR: Jordi Esteve

DEPARTAMENT: Lenguajes y sistemas informáticos

DATA: 27 Enero de 2011

TÍTOL: Lectums: red social para amantes de la lectura

COGNOMS: Hernández Trueba **NOM: Alexandre**

TITULACIÓ: Ingeniería Técnica en Informática

ESPECIALITAT: Gestión **PLA: 92**

DIRECTOR: Jordi Esteve

DEPARTAMENT: Lenguajes y sistemas informáticos

QUALIFICACIÓ DEL PFC

TRIBUNAL

PRESIDENT **SECRETARI** **VOCAL**

DATA DE LECTURA:

Aquest Projecte té en compte aspectes mediambientals: Sí X No

PROJECTE FI DE CARRERA

RESUM (màxim 50 línies)

"Lectums, red social para amantes de la lectura" es un proyecto de fin de carrera con el que se pretende hacer un prototipo para Lectums, una red social dedicada a gente a la que le gusta la lectura y quiere compartir su afición y sus gustos literarios.

Para ello se necesita crear una parte pública en la que los usuarios puedan disfrutar de los servicios típicos de una red social y un extenso catalogo de libros que permitan dotar de contenido a la red. Además, se pretende desarrollar el concepto de biblioteca doméstica, para permitir a los usuarios la gestión de sus libros.

Con el fin de gestionar todo esto, se requiere de un panel de administración relativamente complejo que permita la inserción, modificación y eliminación de todo el contenido, así como la gestión de usuarios.

Todo esto teniendo en cuenta criterios como la accesibilidad o la usabilidad, con el fin de crear una buena experiencia de uso a los usuarios potenciales.

Además, se pretende diseñar, aunque no implementar, una estrategia tanto de negocio, para poder en un futuro sacar rendimiento económico a la red, como de marketing, para lograr que el proyecto tenga el impacto necesario para funcionar.

Paraules clau (màxim 10):

Libro	Web 2.0	Red Social	XHTML
CSS	JQuery	PHP	Codeigniter

Agradecimientos

A mi tutor, Jordi Esteve, por su dedicación al proyecto

A la gente de Stratos AD. Con vosotros comenzó todo

A 37 Signals; vosotros no me conocéis, pero yo a vosotros sí

A mis amigos, porque sin ellos la vida sería tan aburrida…

A Diego, por estar ahí recordándome dónde está la línea

A mi tío, quien me inició en esto de la lectura y me ha apoyado tanto en este proyecto

A mis padres, por haber estado ahí en los buenos y en los malos momentos, cuando más lo necesitaba, por haberme educado, por haberme hecho como soy… sois increíbles

Y a mi novia, Aida, por entender lo que había que entender cuando había que entenderlo, por su cariño, porque ha revolucionado mi vida hasta un punto que no me atrevo a imaginar

Índice

1. El proyecto: Lectums, red social para amantes de la lectura 13
1.1. Introducción 13
1.1.1. Lectums 13
1.1.2. Objetivos 13
1.2. Evaluación tecnológica 14
1.2.1. Opciones tecnológicas disponibles 14
1.2.2. Opción tecnológica escogida: LAMP con “P” de PHP 15
1.2.3. Frameworks disponibles para LAMP 16
1.2.4. MVC: Modelo, Vista y Controlador 17
1.2.5. Otras necesidades tecnológicas 21
1.3. Planificación inicial 22
1.3.1. Análisis de tiempos 22
1.3.2. Estimación económica basada en el análisis de tiempos 25
2. Análisis y especificación 27
2.1. Introducción 27
2.2. Análisis de requisitos 28
2.3. Modelo conceptual 28
2.4. Casos de uso 30
2.4.1. Caso genérico create 32
2.4.2. Caso genérico update 33
2.4.3. Caso genérico delete 34
2.4.4. Caso genérico read 34
2.4.5. Caso finder 35
2.4.6. Caso me gusta 35
2.4.7. Caso no me gusta 36
2.4.8. Caso lo tengo 37
2.4.9. Caso no lo tengo 37
3. Diseño 39
3.1. Introducción 39
3.2. Modelo de componentes 40
3.2.1. Diagrama de componentes 41

3.2.2. Listado de componentes 43
3.3. Diagramas de colaboración 45
3.3.1. Caso genérico: create 45
3.3.2. Caso genérico: read 46
3.3.3. Caso genérico: update 47
3.3.4. Caso genérico: delete 47
3.3.5. Caso: finder 48
3.3.6. Caso me gusta y no me gusta 50
3.3.7. Caso lo tengo y no lo tengo 50
3.4. Diseño de la capa de gestión de datos 51
3.5. Diseño de interfaz 52
3.5.1. Introducción 52
3.5.2. Diseño gráfico 53
3.5.3. Usabilidad 54
3.5.4. Accesibilidad 56
4. Implementación y tests 59
4.1. Introducción 59
4.2. Método clásico 59
4.3. Métodos ágiles 61
4.4. Estrategia de desarrollo 62
4.5. Testing 62
5. Viabilidad 65
5.1. Definiendo la necesidad 65
5.2. Web 2.0 65
5.3. SaaS como modelo de servicio 68
5.4. Idea de negocio 68
5.4.1. Ingresos 68
5.4.2. Gastos 68
5.4.3. Totales 69
5.5. Licencia 69
5.6. Primeros pasos 69
6. Marketing 71

6.1. Introducción 71
6.2. Estrategias SEO 71
6.2.1. ¿Qué es SEO? 71
6.2.2. Técnicas básicas 72
6.2.3. La importancia del contenido 73
6.2.4. Construir buenos enlaces 75
6.2.5. URL's amigables 75
6.2.6. El tamaño de la página web 76
6.3. Publicidad 77
6.3.1. Publicidad gratuita 77
6.3.2. Publicidad de pago 77
6.4. Redes sociales 78
7. Balances y conclusiones 79
7.1. Planificación final y análisis económico 79
7.2. Conclusiones y propuestas de mejora 79
8. Apéndices 83
8.1. Apéndice A: Manuales 83
8.1.1. Manual de usuario 83
8.1.2. Manual de administrador 97
8.2. Apéndice B: Codeigniter, estructura y metodología 102
8.2.1. ¿Qué es Codeigniter? 102
8.2.2. ¿De dónde proviene? 103
8.2.3. ¿Quién utiliza Codeigniter? 103
8.2.4. ¿Y es difícil aprender Codeigniter? 103
8.2.5. ¿Codeigniter va a seguir creciendo? 103
8.2.6. Estructura o Workflow de Codeigniter 104
8.2.7. Metodología habitual 104
8.2.8. Librerías y Helpers 105
8.3. Apéndice C: Software utilizado 105
8.3.1. MacOS 105
8.3.2. Things 106
8.3.3. Coda 107

8.3.4. Transmit 107
8.3.5. MAMP 108
8.3.6. Photoshop 108
8.4. Apéndice D: Hosting web 109
9. Glosario 113
10. Bibliografía 115
10.1. Desarrollo de la lógica 115
10.2. Diseño web 115
10.3. Maquetación 115
10.4. Usabilidad 115
10.5. Accesibilidad 115
10.6. Productividad y negocios 116
10.7. General 116

1. El proyecto: Lectums, red social para amantes de la lectura

1.1. Introducción

1.1.1. Lectums

Lectums será una red social para amantes de la literatura que permitirá gestionar tu **biblioteca personal de libros en la nube**, además de aprovechar los datos de los demás usuarios y los tuyos propios para disponer de un sistema de **recomendación de libros** nunca antes visto. Realmente se trata de ayudar al usuario a encontrar nuevos libros para leer que coincidan plenamente con sus gustos e inquietudes, además de gestionar su propia **colección de libros doméstica**. También intenta ser un punto de inicio para la compra de libros a precios competitivos, en función, quizás, de la localización del propio usuario; dependiendo de la disponibilidad de los datos, se podría estudiar añadir la opción de buscar los libros en las bibliotecas cercanas.

Lectums es una gran base de datos de libros, construida por el administrador. Éste uno de los puntos fuertes de **Lectums**, pues el contenido generado es muy basto y permite una buena indexación, en principio, de los buscadores, aspecto capital en un proyecto online.

Especial atención al concepto de biblioteca doméstica: se deberían implementar el mayor número de servicios útiles para que los usuarios pudieran entender **Lectums** como un lugar útil incluso sin el concurso de nadie más, es decir, incluso sin las ventajas del concepto de red social. Podemos observar **Delicious Library**, un buen gestor de libros para **MacOS** con multitud de servicios interesantes.

Es muy importante hablar con las principales tiendas de libros online para poder negociar acuerdos interesantes que permitan obtener descuentos para los usuarios de **Lectums**, así como para integrarlas en el modelo de negocio. Sobre dicho modelo, hay que añadir los posibles ingresos por publicidad, lo que hace necesario obtener un buen tráfico de visitas desde el principio.

Se deberá diseñar y programar el panel de administración adecuado y suficiente para gestionar el contenido de la red social. En concreto, debe permitir introducir, modificar y eliminar todo el contenido bibliográfico, así como gestionar los usuarios y otros contenidos más transversales, como las FAQ.

1.1.2. Objetivos

- Crear una red social centrada en la lectura. Esto incluye:
    - Una parte pública en la que los usuarios puedan ver información sobre libros.
    - Una parte pública en la que los usuarios puedan administrar su perfil.

    - Una parte pública en la que los usuarios puedan administrar su biblioteca personal.
    - Una parte pública en la que los usuarios puedan interaccionar.
    - Un administrador para poder gestionar la red.
- Diseñar una política de marketing adecuada para promocionar la red social y aumentar su impacto.
- Diseñar una estrategia para poner en práctica el plan de negocio.

1.2. Evaluación tecnológica

1.2.1. Opciones tecnológicas disponibles

1.2.1.1. LAMP: con "P" de PHP o con "P" de Python

LAMP son las siglas de Linux, Apache, MySQL y PHP o Python. Es decir, se trata de una infraestructura basa en el sistema operativo libre, Linux, un servidor también de código abierto, Apache, y una base de datos MySQL, de Oracle. Como lenguaje de programación, LAMP tiene dos opciones: PHP o Python, ambos lenguajes de script perfectamente conocidos en Internet. Estudiemos brevemente cada una de las tecnologías:

- **Linux:** sistema operativo creado por Linus Torvalds que se basa en el nucleo de Unix y utiliza las herramientas del proyecto GNU –motivo por el que se conoce a Linux como GNU/Linux- para la interacción con el usuario. Es un sistema que se basa en su mayor parte en software libre, con lo cual los desarrolladores pueden acceder al código fuente, en principio para aprender y colaborar en su desarrollo.

  En la práctica Linux se ha convertido en el sistema operativo más usado para servidores y sistemas de red, aunque tiene poco éxito en el mercado del usuario final, pese a la insistencia de instituciones públicas como las propias Universidades debido, probablemente, a su insuficiente usabilidad. Pese a ello, nadie duda de la calidad de GNU/Linux, que destaca por su seguridad frente a otros sistemas operativos como Microsoft Windows.
- **Apache:** de la Apache Software Foundation, es el servidor *http* más utilizado del mundo. Es software libre y destaca por su seguridad y facilidad de uso, así como por su buena integración en Linux que, como decía anteriormente, es el sistema operativo más frecuente en entornos de servidor. Soporta varios lenguajes de programación, entre ellos PHP y Python.
- **MySQL:** es probablemente el gestor de bases de datos más utilizado en Internet, al menos para proyectos standard. No es la mejor opción para proyectos de requerimientos extraordinarios, como podría ser por ejemplo la red social Facebook, que utiliza un sistema de bases de datos no relacional propio. No obstante, sí cumpliría con las necesidades de Lectusm y, por tanto, es una opción a tener en cuenta.
- **PHP:** el lenguaje de scripting para Internet más popular, destaca por su curva de aprendizaje reducida y la cantidad de información y recursos que existen de forma gratuita en Internet. Pese a tener importantes carencias –como la sobrecarga-, que además no parece que se vayan a solucionar

próximamente debido a que hay ciertos conflictos sobre el camino que seguirá PHP en el futuro, lo cierto es que un lenguaje bueno con una gran aceptación.

- **Python:** este lenguaje sigue las técnicas modernas, en cuanto a que simplifica la sintaxis y permite patrones y métodos novedosos en el desarrollo de software. Es muy popular, especialmente a nivel didáctico, pues pese a que evita algunas restricciones semánticas habituales, como el punto y coma a final de línea y obliga a respetar la identación en el código, algo muy importante para el mantenimiento posterior del código. Bastante popular en el mundo web, aparece normalmente asociado a algún framework de desarrollo, como Django, y pocas veces *en estado puro*. La empresa que más ha hecho, probablemente, por popularizar este lenguaje es Google, que lo utiliza habitualmente en sus desarrollos. Se dice, no obstante, que lo está abandonando en favor de Java.

1.2.1.2. RoR

RoR son las siglas de Ruby on Rails –Ruby sobre raíles-, y dan nombre al framework de desarrollo basado en Ruby que se ha hecho popular –y que ha popularizado el propio lenguaje Ruby- en los últimos años, gracias, sobre todo, al trabajo de 37Signals con Basecamp. De hecho, para ser sinceros, Ruby no es un lenguaje muy usado por separado, pese a ser un gran lenguaje que permite novedosas técnicas como la metaprogramación. En honor a la verdad, casi todos los frameworks de desarrollo que han aparecido para otros lenguajes se han basado en **Ruby on Rails** y, debido a las limitaciones de sus respectivos lenguajes de programación, muy pocos –o ninguno- han llegado a su nivel. Es un framework de desarrollo muy bueno que, pese a tener sus desventajas, aumenta la productividad enormemente, permitiendo desarrollos muy rápidos.

1.2.1.3. .Net

.Net es la apuesta de Microsoft para el desarrollo en general. En el caso de la web, su propuesta ASP.NET es bastante conocida y se basa en su arquitectura de servidores IIS –Internet Information Services. Pese a contar con el apoyo de Microsoft, lo cierto es que no ha alcanzado la popularidad y el nivel de uso esperado. No obstante, es una opción muy frecuente en determinados sectores, y sigue con el éxito que tuvo ASP –que se parecía mucho a PHP- en su momento. Además, cuenta con la ventaja de desarrollarse con Visual Studio que es, a mi modo de ver, el mejor IDE que existe en el mercado.

1.2.2. Opción tecnológica escogida: LAMP con "P" de PHP

Después de estudiar con detenimiento las opciones que tenía, la escogida ha sido **LAMP con P de PHP**. Hay muchos motivos detrás de esta decisión, entre los cuales destaco los siguientes:

- Muy frecuente en el mundo del desarrollo web, la documentación es infinita a lo largo de la red.
- Como es muy frecuente, el número de servicios de hospedaje que soportan esta tecnología es virtualmente ilimitado, y las ofertas muy buenas –a diferencia de, por ejemplo, **RoR**.
- Cuento con una experiencia de varios años en el desarrollo de páginas web con esta tecnología, lo que me debería permitir desarrollar **Lectums** con todas las garantías.

La otra opción que más fuerza tenía para mí era **RoR**, porque lo cierto es que el framework es sublime; no obstante, mi mayor experiencia en entornos **LAMP** empujó la balanza hacia este último, y considero que cualquiera de las opciones expuestas era suficientemente buena.

1.2.3. Frameworks disponibles para LAMP

1.2.3.1. CakePHP

CakePHP es un framework para el desarrollo de aplicaciones web en **PHP** basado en los paradigmas **MVC (Model, View, Controller)** y **ORM (Object Relational Mapping).** El primero será explicado en el siguiente subapartado; respecto a **ORM**, consiste en relacionar los modelos de la aplicación con las tablas equivalentes de la base de datos: el propio modelo representaría la tabla, cada instancia del modelo un registro, y sus atributos representarían los campos de dicha tabla. Al utilizar metaprogramación, los modelos se "metamorfosean" convirtiéndose en fieles reflejos de las tablas de la base de datos. Esto se lleva al extremo en **RoR**, y se trata de una forma más superficial en **CakePHP** y prácticamente inexistente en **Codeigniter**; el hecho de que **PHP** disponga de una peor orientación a objetos que **RoR** es sin duda la mayor motivación.

Una de las señas de identidad de **CakePHP** es la práctica ausencia de configuración. En otros frameworks de desarrollo, especialmente en **Java**, se realiza un mapeo de las tablas en **XML** que hay que hacer a mano y que puede resultar tedioso. En **CakePHP**, como en **RoR**, se utilizan unas convenciones de nombres que eliminan la necesidad de este mapeo.

La licencia bajo la que está distribuido **CakePHP** es la **MIT**, lo cual da una libertad total para desarrollar en él.

1.2.3.2. Codeigniter

Codeigniter es probablemente uno de los frameworks más ligeros para PHP. Sin llegar a ser tan potente en cuanto a **ORM** como **CakePHP** o –ni mucho menos- **RoR**, es un framework que permite un desarrollo muy ágil y sencillo. Su curva de aprendizaje es ostensiblemente menos pronunciada que las de los otros dos, lo cual le está convirtiendo en un framework de referencia. Al momento de escribir estas líneas se encuentra sumido en un proceso de cambio desde la versión 1.7.3 a su otra rama, la 2, que pasará a ser la oficial. **Codeigniter** nace del **CMS ExpressionEngine**, que pertenece a **Ellis Lab**; sin embargo, **Codeigniter** se distribuye bajo licencia **Apache/BSD**.

Algunos estudios de rendimiento indican que **Codeigniter** es sustancialmente más rápido que **CakePHP**, lo que le convertiría en el framework más rápido entre los frameworks maduros para **PHP**. Igual que en el caso de **CakePHP**, la configuración necesaria es mínima. Además, en el caso de **Codeigniter**, soporta un abanico enorme de hostings posibles. Esta es una gran ventaja no

tanto sobre **CakePHP**, que es más o menos igual en este aspecto, sino sobre **RoR**, que requiere un servidor con soporte para **Ruby**, lo cual hasta hace poco no ha sido nada frecuente.

Se hablará en más profundidad de Codeigniter en los siguientes apartados y, especialmente, en el Apéndice B.

1.2.3.3. Framework escogido: Codeigniter

Finalmente el framework escogido ha sido **Codeigniter**, puesto que a sus características hay que sumarle mi experiencia en su uso que supera los 4 años. En este caso, **CakePHP** o **Codeigniter** eran opciones perfectamente viables, ambas, y ha sido mi comodidad con este último lo que ha inclinado la balanza.

Se hubiera podido desarrollar **Lectums** en **CakePHP** sin ningún problema, aunque la curva de aprendizaje hubiera sido mayor y, por tanto, el tiempo de desarrollo también. En caso de haber escogido como lenguaje **Ruby**, **RoR** hubiera sido la única opción viable, pero su curva de aprendizaje, la más pronunciada, hubiera podido comprometer la viabilidad del proyecto en los tiempos con los que contaba.

1.2.4. MVC: Modelo, Vista y Controlador

1.2.4.1. Introducción y origen

MVC son las siglas de Modelo, Vista, Controlador y es un paradigma de desarrollo de software que no surge, desde luego, a partir de los frameworks de desarrollo para la web. Fue descrito por primera vez en 1979 por Trygve Reenskaug[1], que entonces trabajaba en Smalltalk en los laboratorios de Xerox, y se centraba en el sistema de multiventanas de dicho lenguaje de programación.

Trygve Reenskaug hablaba de sistemas como el Apple Lisa, los Macintosh y "otros muchos imitadores", para los que estaba destinado este sistema de interfaces. Cada una de las múltiples ventanas que podía tener una interfaz estaba controlada por una triada MVC o, dicho de otra manera, por un controlador, un modelo y una vista. La vista, por supuesto, se encarga de mostrar en pantalla la salida gráfica o textual correspondiente en la porción de pantalla destinada a ello en esa ventana; el controlador se encarga de recoger los eventos del mouse o del teclado y transmitirles los cambios oportunos al modelo y la vista; el modelo, por último, gestiona el entorno y los datos, responde a las peticiones sobre el estado, especialmente de la vista, y actúa ante las peticiones de cambio de estado del controlador. Existía una relación fuerte entre el controlador y la vista, pues sólo podía haber una vista por cada controlador, pero podía haber

[1] Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)

varios modelos actuando sobre una misma vista –porque de hecho era perfectamente posible que las vistas solicitaran datos o estados de varios modelos diferentes para formar una única vista.

Hoy en día pocos –o ningún– sistema de interfaz utilizan esta triada MVC, sino que han evolucionado a otros sistemas que prometen ser más eficientes en este campo; sin embargo, MVC se ha popularizado en el desarrollo web donde, planteado de una manera sensiblemente diferente, ha dado resultados muy buenos y es clave en la proliferación de frameworks de desarrollo web como CodeIgniter.

1.2.4.2. MVC en el desarrollo web

En una aplicación web la gestión del input es responsabilidad del navegador en tanto en cuanto los elementos con los que se puede interactuar son controlados por éste en su inmensa mayoría –hablo de campos de formularios, enlaces, etc. Por este motivo, el controlador planteado en la época de smalltalk no tendría ningún sentido en la actualidad, al no ser necesaria la gestión del input.

Como la web es en cierta manera pasiva –en condiciones normales requiere de un cambio o actualización de página, esto es, una nueva petición al servidor, para modificar los datos que se muestran– tampoco tiene especial sentido que un controlador sólo tenga asociada una vista, pues sería imposible, en ese caso, mostrar diferentes datos con un mismo controlador –algo perfectamente deseable– al tener sólo una vista disponible que además no puede modificarse.[2]

Por lo tanto queda claro que el paradigma MVC tal como se planteó hace treinta años no tiene demasiado sentido en el campo del desarrollo web y, en este punto, hay que darles la razón a los que consideran que este patrón de diseño no es adecuado para la web[3]. No obstante, los frameworks de desarrollo web en realidad no utilizan MVC tal cual, sino que lo utilizan de una manera bastante diferente que, bajo mi punto de vista, sí tiene un gran sentido y facilita el desarrollo de aplicaciones de cierto tamaño.

¿Y en qué consiste este uso? Pues en realidad es casi una influencia, una inspiración del sistema original, que conserva poco más que el nombre y los objetos fundamentales. Primeramente, en el MVC utilizado por la mayoría de frameworks un controlador puede llamar a cuantas vistas quiera aunque, eso sí, el resultado de ello es una única vista que las comprende todas, puesto que el navegador sólo puede mostrar una página a la vez. Por lo tanto ya no existe esa relación fuerte

[2] Ajax es, por supuesto, una pega en la argumentación, puesto que permite modificar la apariencia de las páginas –o vistas– sin necesidad de cambiar de página o de actualizarla, al realizar una petición transparente que da cierto grado de dinamismo a las aplicaciones web acercándolas, en cierta manera, a lo que esperamos de una aplicación de escritorio. No obstante, aquí hablo del caso habitual.

[3] Esto no debería sorprenderle a nadie teniendo en cuenta que en esa época ni siquiera existía la World Wide Web.

entre controladores y vistas. En cuanto a los controladores en sí, ya no gestionan el input, pues esto no tiene sentido, sino que ahora se dedican a controlar las peticiones del navegador o, dicho de otra forma, a gestionar las URL's; ya no es necesario un archivo por cada URL que exista en nuestro sitio web sino que, en vez de eso, un controlador, que es un único archivo, puede gestionar un número virtualmente ilimitado de URL's que, obviamente, muestran vistas diferentes. Además, el controlador se encarga de pedir los datos al modelo y enviárselos a la vista para que pueda formatear la salida adecuadamente.

Como ya se puede intuir, los modelos sí conservan su uso inicial pero, en caso de estar estos en una base de datos, generan –o deberían generar– una estrecha relación entre el propio modelo y la tabla de la base de datos que corresponda al concepto que encierra ese modelo.

1.2.4.3. Modelo conceptual (modelo entidad-relación)

Como MVC no se está usando para gestionar ventanas, ya no hay una relación entre una triada de modelo, vista y controlador y una ventana en concreto; lo que se intenta ahora es asociar modelos y controladores con entidades concretas de un modelo entidad-relación.

Un modelo entidad-relación intenta representar conceptos de un problema de software y las relaciones que existen entre éstos. Estrictamente hablando no existe una relación formal entre el modelo entidad-relación y la programación orientada a objetos pero, de hecho, la programación orientada a objetos es una buena manera de implementar soluciones basadas en este modelo. Tanto los controladores como los modelos son objetos en los frameworks de desarrollo para la web, por lo que su traducción del modelo entidad-relación a la implementación es trivial.

Suponiendo que quisiéramos implementar el clásico blog en MVC, podríamos disponer de un modelo entidad-relación que tuviera las siguientes entidades: post, autor y comentario[4]. Las relaciones entre estos consistirían en que un autor escribe un post y que en un post posee comentarios.

Una forma muy sencilla de convertir estas entidades en controladores, modelos y vistas es crear un controlador para cada una de las entidades sobre las que vayan a haber acciones: crearemos posts, por lo tanto *post* tiene que ser un controlador; crearemos comentarios, por lo tanto *comentario* tiene que ser un controlador; si quisiéramos gestionar varios autores, *autor* tendría que ser un controlador. En el caso de los modelos, crearíamos un modelo para cada entidad que vaya a tener datos en una base de datos, en este caso todos. Para las vistas, crearemos una vista para cada de una de las acciones de cada uno de los controladores.

[4] Por supuesto estoy simplificando enormemente la estructura de un weblog real.

Esta "regla" no es infalible y debe ser tratada con cuidado pero, en general, si tu modelo entidad-relación es suficientemente completo, no crearás nada que sobre y no dejarás nada por crear.[5]

1.2.4.5. ¿Por qué es importante el modelo conceptual?

A estas alturas alguien podría preguntarse por qué es importante diseñar un modelo conceptual en vez de ponerse directamente manos a la obra y crear controladores y modelos en función de las necesidades del desarrollo.

No soy partidario del exceso de planificación en un proyecto de software porque creo sinceramente que en muchos casos actúa como freno y hace perder oportunidades de éxito en ideas que dependan del tiempo. No obstante en este caso el modelo conceptual es básico porque permite no divagar, no improvisar. Un modelo conceptual permite diseñar toda la estructura del sitio web en poco tiempo sin necesidad siquiera de implementar el contenido de controladores o modelos. Permite, de hecho, el trabajo en grupo, al poderse crear toda la estructura de ficheros y dividir la tarea entre varios programadores; estos últimos sólo tienen que rellenar los métodos de cada controlador o modelo, mientras algunos maquetadores se encargan de crear las vistas.

No me gusta, sin embargo, crear diagramas de componentes o de colaboración para el desarrollo con un framework web salvo para métodos especialmente complejos. El desarrollo web habitual se compone de muchísimos métodos triviales –borrar un post, por ejemplo, no requiere un estudio de ingeniería– con lo que es de hecho más rápido implementarlo directamente que diseñarlo primero y luego implementarlo.

1.2.4.6. Las URL's: hablamos de métodos

Antes comentaba que es posible gestionar varias URL's diferentes con un mismo controlador. ¿Cómo es esto posible? Muy sencillo. Al ser los controladores objetos –igual que los modelos– cada controlador es una clase y, por tanto, tiene atributos y métodos. Cada método es una acción que se produce sobre el concepto representado por el controlador y la forma de ejecutar es acceder mediante una url; de esta forma, construiremos nuestras URL's de la siguiente manera:

dominio.com/controlador/método

Al acceder a esa url, se ejecutaría el método correspondiente a ese controlador; si siguiéramos con nuestro ejemplo de antes y quisiéramos ver los posts de nuestro blog, podríamos crear un método en el controlador "post" que se llamara "ver", y lo llamaríamos mediante la siguiente URL:

[5] De hecho en el 99% de los casos cada entidad es un controlador, a diferencia de los modelos que pueden existir o no dependiendo de si la entidad requiere guardar datos o no. Las excepciones suelen consistir en aquellos casos en que la entidad responde a una necesidad semántica muy especial que sólo tiene relevancia informativa.

miblog.com/post/ver

Esta es una manera estupenda de trabajar porque no sólo provee una estructura muy lógica sino que además proporciona URL's con mucha más semántica al incorporar elementos que en realidad no existen –no hay una carpeta llamada "post" y un archivo llamado "ver".

¿Qué pasa si queremos ver un *post* en concreto, digamos, el que tiene como identificar el número 23? En este caso, echamos mano de los parámetros de los métodos; añadiríamos el parámetro "id" al método ver, y accederíamos a nuestro *post* con la siguiente URL:

miblog.com/post/ver/23

Podemos, de hecho, incorporar cualquier número de parámetros a un método, ya que las URL's entienden –mediante un enrutador que explicaremos más adelante y que es diferente para cada framework– que el primer elemento de la URL después del dominio es el controlador, el segundo el método y, todos los demás, son atributos. Podríamos, por tanto, establecer como parámetros de nuestro método "ver" "título" y "id" de forma que accediéramos así:

miblog.com/post/ver/mi-primer-post/23

Muy interesante porque aunque en realidad el identificador 23 es suficiente para encontrar el *post* correcto, añadir el título a la URL añade información semántica muy útil tanto para personas como para buscadores.

1.2.5. Otras necesidades tecnológicas

Evidentemente hay una serie de necesidades tecnológicas que van más allá de la programación de la lógica de aplicación desde el lado del servidor. Concretamente estoy hablando de la presentación y la lógica desde el lado cliente, para lo que es necesario –entre otras cosas- **XHTML**, **CSS** y **JavaScript**. Para este último se ha utilizado por motivos de productividad y posibilidades técnicas el framework **JQuery**.

1.2.5.1. XHTML

XHTML es el lenguaje de marcas estándar potenciado por la **W3C** y que sustituye a **HTML 4**. La idea es sentar unas reglas más estrictas que permitan una mayor estandarización de la web y que aplique conceptos semánticos novedosos. De hecho, convierte las antiguas etiquetas de **HTML 4** en etiquetas **XML**, **de aquí la "X" de XHTML.**

Si bien es cierto que **HTML5** ya apunta con fuerza, también lo es que los navegadores de una parte básica de los visitantes de cualquier página web aún no soportan casi ninguna de las nuevas etiquetas y métodos, por lo que desarrollar todavía para **HTML5** es inviable, salvo que proveas

opciones alternativas. En el caso de **Lectums**, he considerado que aún es demasiado pronto y que tampoco hay puntos conflictivos que requirieran una solución más sofisticada, por lo que no he aplicado **HTML5**.

Se ha intentado utilizar **XHTML** estricto, de forma que valide en los controles de validación del **W3C**.

1.2.5.2. CSS

CSS (Cascading StyleSheet) es un lenguaje de estilos que permite formatear la salida **XHTML**. Es el estándar reconocido por el **W3C** y la única opción viable en cuanto al diseño de las páginas web –más allá de imágenes que enriquezcan la maqueta, por supuesto. Se ha intentado utilizar etiquetas compatibles con la mayoría de navegadores y, en caso de no poder, ha sido en casos en que ello no repercutía en la experiencia del usuario.

1.2.5.3. JavaScript (JQuery)

JavaScript es un lenguaje desarrollado inicialmente por Netscape que permite programar la lógica de la aplicación desde el lado del cliente. Es enormemente útil y en los últimos años es el responsable del cambio de concepto de la web que la ha acercado a las aplicaciones de escritorio tradicionales.

JQuery es un framework que permite el uso intensivo y sencillo de **JavaScript**, así como el uso de efectos muy interesantes que hacen de la aplicación web algo vivo. La capacidad de utilizar **AJAX** –tecnología que permite enviar peticiones al servidor sin cambiar de página- ha permitido que en los últimos tiempos el comportamiento habitual de una página web, esto es, efectuar una opción y esperar una respuesta por parte del servidor que implicaba necesariamente un cambio de página, cambie totalmente.

1.3. Planificación inicial

1.3.1. Análisis de tiempos

1.3.1.1. Introducción

Dada la complejidad del proyecto es fundamental la planificación y el análisis de tiempos; en el caso de que el proyecto fuera dirigido por una empresa, esta planificación y el análisis de cómo se ha implementado marcaría si el proyecto es rentable o no, incluso siendo posible su cancelación si en algún punto del proyecto se constata la imposibilidad de cumplir los plazos y el descenso o eliminación de la rentabilidad.

Es evidente que un proyecto de esta envergadura involucra a distintos profesionales. En mi caso, al ser el único disponible, ejecutaré el papel de todos ellos yo mismo; no obstante, hay que especificar en qué proporción actuaré como cada uno de ellos.

1.3.1.2. Analista

El analista es el responsable del correcto diseño de la aplicación[6]. Debe responder a las siguientes preguntas:

- ¿Cuál es el problema o la necesidad del cliente?
- En función de ello, ¿qué funcionalidades requiere la aplicación y cómo se han de llevar a cabo?

Aquí yo tengo cierta ventaja porque además del analista soy en cierta manera el cliente; esto hace que tenga muy claras las necesidades del proyecto y que su diseño no sea tan complejo como el de un proyecto encargado por un tercero. Otra ventaja es el uso del framework Codeigniter, que al ser un framework MVC, marca mucho el diseño conceptual de la aplicación. Un buen modelo conceptual en la especificación marca absolutamente el diseño, hasta el punto de ser prácticamente automático (una entidad es un controlador y en muchos casos un modelo, cada caso de uso es un método y una vista…).

A esta tarea, por tanto, preveo dedicarle alrededor de 75 horas.

1.3.1.3. Programador

El programador debe llevar a cabo el diseño elaborado por el analista sin salirse de la línea marcada excepto en las decisiones propias del lenguaje. Es uno de los profesionales que más trabajo tiene en este caso, porque debe comprender el diseño e implementarlo y, en este caso, no son pocas las funcionalidades a implementar.

Para esta tarea preveo 150 horas de trabajo.

1.3.1.4. Maquetador

El maquetador implementa en XHTML y CSS el diseño creado por el diseñador web, respetando en cualquier caso las reglas de accesibilidad exigibles. Entre ambos han de tener en cuenta la usabilidad.

Para esta tarea preveo emplear 100 horas.

[6] Evidentemente no me estoy refiriendo a diseño gráfico, sino al diseño conceptual –según la Ingeniería del Software.

1.3.1.5. Diseñador web

El diseño[7] es fundamental para el éxito del proyecto, ya que responde las siguientes preguntas:

- ¿Cómo debe visualizar el usuario la aplicación?
- ¿Cuál es la mejor manera de que el usuario entienda el propósito de la misma y se anime a interactuar?
- ¿Cuál es el mejor diseño de interacción?

Todas estas preguntas requieren de un estudio previo tanto del mercado como de la audiciencia objetivo.

Para esta tarea preveo emplear 60 horas.

1.3.1.6. Documentador

Es fundamental documentar la aplicación, en este caso, haciendo la memoria. Creo que dedicaré aproximadamente 50 horas a ello.

1.3.1.7. Introductor de contenido

Es muy importante que **Lectums** aparezca con contenido para posicionar correctamente y aportar valor al usuario. Planeo emplear 100 horas.

1.3.1.8. Totales

Profesional	**Horas**	**Total**
Analista	75	75
Programador	150	225
Maquetador	100	325
Diseñador web	60	385
Documentador	50	435
Introductor de contenido	100	535

Tabla 1 Horas totales

[7] Ahora sí hablo de diseño gráfico, aunque también podemos hablar de diseño de la usabilidad, de la interfaz, del entorno, etc.

1.3.2. Estimación económica basada en el análisis de tiempos

1.3.2.1. Introducción

A continuación quiero realizar un cálculo económico del impacto del proyecto. Me voy a basar en los salarios mínimos de los distintos profesionales, aunque es evidente que esto no va a ser así en la mayoría de los casos; supondré que hay que añadir un 35% más por gastos de Seguridad Social a cargo de la empresa y otro 35% por gastos comunes (agua, luz, gas…). Suponemos 1760 horas útiles, esto es, 160 horas * 11 meses (1 son vacaciones).

Profesional	**Precio/Hora**	**Horas**	**SS**	**Varios**	**Subtotal**	**Total**
Analista	25,56	75	8,95	8,95	3334,5	3334,5
Programador	19,9	150	6,97	6,97	5076	8410,5
Maquetador	19,9	100	6,97	6,97	3384	11794,5
Diseñador web	19,9	60	6,97	6,97	2030	13824,5
Documentador	11,4	50	4	4	970	14794,5
Introductor de contenido	11,4	100	4	4	1940	16734,5

Tabla 2 Costes totales

Evidentemente estos son gastos realistas teniendo en cuenta costes reales de mercado. A medida que aumenta el tamaño de la empresa este coste sube, llegando al millón de euros o más en algunos proyectos similares realizados por grandes consultoras. En el caso de la Administración Pública, este coste podría llegar a ser ridículamente superior.

2. Análisis y especificación

2.1. Introducción

Como se ha dicho en la introducción, **Lectums** pretende ser la red social de referencia para los amantes de la lectura. Esto implica una serie de necesidades básicas que ha de cumplir, así como una serie de "extras" que debería tener si quiere tener un mínimo éxito. Paso a definir a continuación las que creo son las claves del proyecto. Hay que tener en cuenta que los plazos dados para hacer un proyecto de fin de carrera no son suficientes para desarrollar por completo un producto de esta envergadura, con lo que se ha de entender el resultado de este proyecto y sus necesidades como un prototipo desde el que se pueda crear el producto final.

Lectums tiene una necesidad básica: una base de datos de información sobre libros con unas características concretas: título, autor o autores, una descripción, una o varias categorías y una o varias ediciones que especifiquen la editorial que las ha editado. Además, los libros pueden pertenecer a colecciones, que podrían asociar los libros por saga, mundo sobre el que transcurren, etc. Debido a la relación inalienable entre el contenido y el posicionamiento en las buscadores, así como el propio interés de los usuarios, esta base de datos debería ser lo más grande posible. Para ello, es necesaria una herramienta, el panel de administración, que permita una actualización fácil de la base de datos.

Como cualquier red social, es necesario que los usuarios dispongan de una serie de herramientas que les permitan comunicarse. En concreto, se necesita una página de perfil, con la información del usuario: nombre, apellidos, nombre de usuario, correo electrónico, qué libros está leyendo en este momento, cuáles son sus libros favoritos, y cuáles tiene. Además, ha de poder enviar mensajes públicos a otros usuarios, de forma que pueda existir una conversación pública similar al muro de Facebook. Por último, los usuarios han de poder introducir comentarios en varias áreas de la web, en concreto en las páginas de libros.

El panel de administración ha de permitir gestionar los usuarios, de forma que se puedan borrar los usuarios no válidos, crear usuarios manualmente, editarlos, etc.

Antes se comentaba que los usuarios tienen listados de libros que poseen. Esto desemboca en el concepto de biblioteca. La biblioteca es una base de datos personal en la que cada usuario tiene almacenados los libros de que dispone, de forma que puede especificar dónde los tiene –en una estantería, en un armario…– y puede registrar si se los ha prestado a alguien y quién es, lo que podría solucionar el frecuente problema de la pérdida de un libro por no saber a quién se lo has prestado.

Como en cualquier otra aplicación web, se requieren una serie de servicios mínimos. En concreto, un sistema de FAQ's actualizable desde el panel de administración, así como algunas páginas estáticas, la de contacto, la de *Qué es Lectums*, etc.

2.2. Análisis de requisitos

De la introducción de la especificación se desprenden una serie de requisitos del sistema. Estos requisitos se han de tomar de forma genérica, pues será después al definir los casos de uso cuando se especifiquen de forma más precisa. Los requisitos detectados son los siguientes:

- Bases de datos
    - Libros
    - Ediciones
    - Categorías
    - Colecciones
    - Autores
    - Editoriales
- Usuarios
    - Se han de poder crear usuarios.
    - Los usuarios han de poder acceder al sistema.
    - Se requiere una página de perfil con los datos básicos de los usuarios.
    - Se ha de poder gestionar toda la información relativa a los usuarios.
    - Los usuarios han de poder indicar los libros que les gustan.
    - Los usuarios han de poder indicar qué libros tienen.
    - Los usuarios han de poder especificar qué libros están leyendo actualmente.
    - Un usuario ha de poder enviar un mensaje público a otro.
    - Un usuario ha de poder introducir un comentario sobre un libro.
- Biblioteca
    - Un usuario ha de poder especificar dónde está cada uno de los libros que tiene en su biblioteca.
    - Un usuario ha de poder especificar si alguno de sus libros está prestado, y a quién le ha hecho el préstamo.
- Extras
    - FAQ's

2.3. Modelo conceptual

A partir de la introducción a la especificación y de la lista de requisitos, se genera el siguiente modelo conceptual. Hay que tener en cuenta que este modelo se refiere al problema, es decir, a la

situación antes de aportar la solución. Obviamente, algunas de las cosas que aparecen en el modelo después no tendrán sentido en la implementación, por poderse ignorar o ser simples vistas o consultas a bases de datos. En resumen, el modelo conceptual da una visión global de la situación planteada, no de la forma de lograrla.

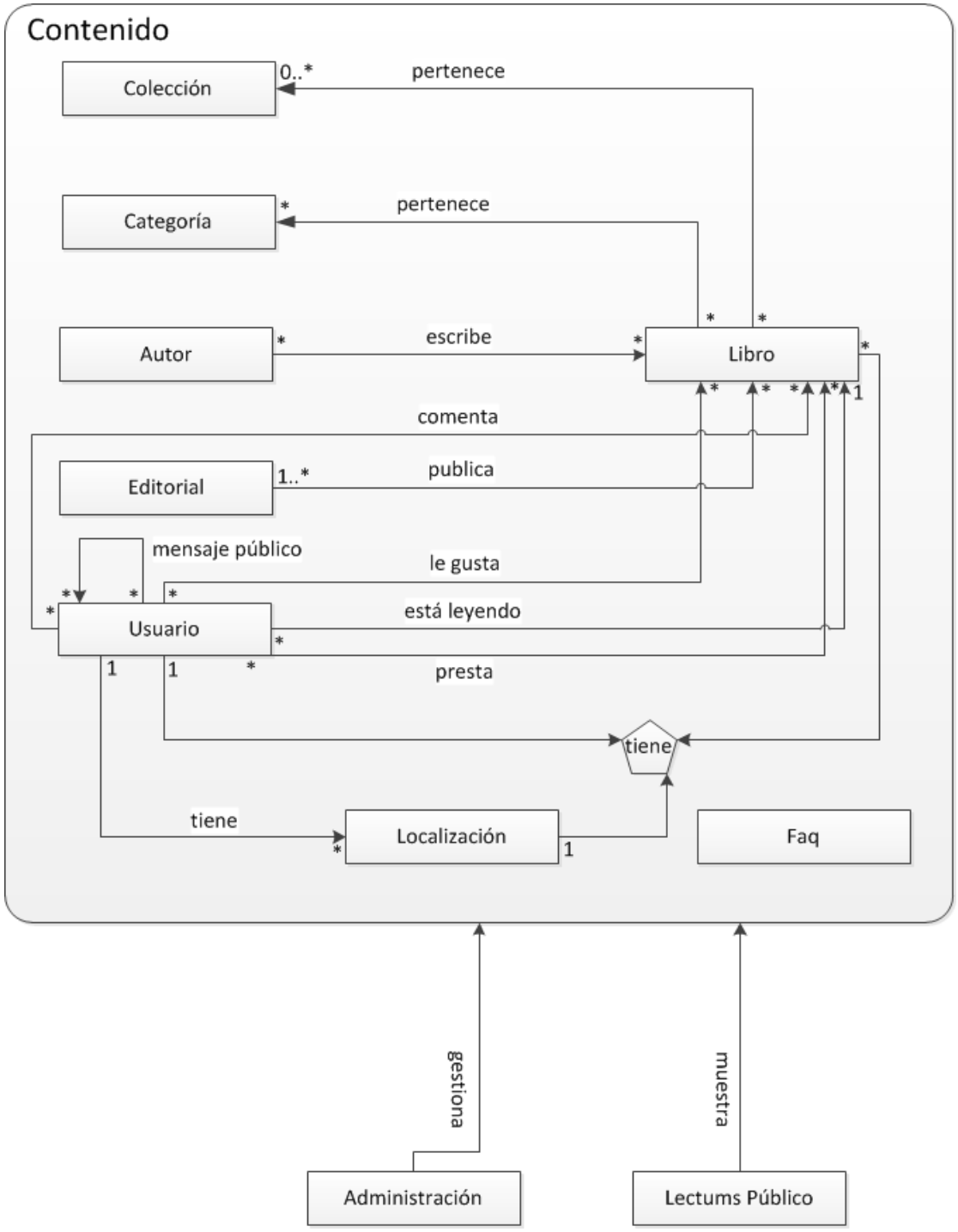

Ilustración 1 Modelo conceptual

2.4. Casos de uso

A continuación se expone la totalidad de casos de uso del proyecto de una forma genérica. Sólo se explicarán en profundidad los casos de uso que sean relativamente complejos, y simplemente se nombrarán aquellos que sean triviales. Cada caso de uso pertenece, obviamente, a un requisito; para diferenciar los casos de uso de los requisitos generales, estos últimos están en cursiva.

- Bases de datos
    - Libros
        - *Listar todos*
        - *Ver libro*
        - *Buscar libro*
        - *Crear libro*
        - *Modificar libro*
        - *Borrar libro*
        - *Añadir ítem a un libro (un autor, una colección, una categoría…)*
        - *Quitar ítem de un libro (un autor, una colección, un categoría…)*
    - Categorías
        - *Listar todas*
        - *Ver libros de una categoría*
        - *Crear categoría*
    - Colecciones
        - *Listar libros de una colección*
        - *Crear colección*
        - *Buscar colección*
    - Autores
        - *Ver autor*
        - *Listar autores*
        - *Listar libros de un autor*
        - *Crear autor*
        - *Buscar autor*
    - Editoriales
        - *Listar editoriales*
        - *Listar libros de una editorial*
        - *Crear editorial*
        - *Buscar editorial*

- Usuarios
    - Los usuarios han de poder acceder al sistema y salir de él.
        - *Login*
        - *Logout*
    - Se requiere una página de perfil con los datos básicos de los usuarios.
        - *Ver perfil*
        - *Modificar perfil*
    - Se ha de poder gestionar toda la información relativa a los usuarios.
        - *Crear usuario*
        - *Listar usuarios*
        - *Modificar usuarios*
        - *Borrar usuario*
    - Los usuarios han de poder indicar los libros que les gustan.
        - *Me gusta*
        - *Ya me gusta*
    - Los usuarios han de poder indicar qué libros tienen.
        - *Lo tengo*
        - *Ya no lo tengo*
    - Los usuarios han de poder especificar qué libros están leyendo actualmente.
        - *Modificar libros actuales*
    - Un usuario ha de poder enviar un mensaje público a otro.
        - *Enviar mensaje públicos*
        - *Ver mensajes públicos*
    - Un usuario ha de poder introducir un comentario sobre un libro.
        - *Comentar libro*
- Biblioteca
    - Un usuario ha de poder especificar dónde está cada uno de los libros que tiene en su biblioteca.
        - *Crear localización*
        - *Modificar localización*
    - Un usuario ha de poder especificar si alguno de sus libros está prestado, y a quién le ha hecho el préstamo.
        - *Activar préstamo*
        - *Cancelar préstamo*

- Extras
    - FAQ's
        - *Listar FAQS*
        - *Crear FAQ*
        - *Modificar FAQ*
        - *Borrar FAQ*

A continuación, explico algunos casos de uso que considero interesantes; los cuatro primeros, create, update, delete y read son casos genéricos. En Lectums hay decenas de creates, updates, deletes y reads y, por tanto, resultaría improductivo explicarlos todos uno por uno siendo estos prácticamente iguales.

2.4.1. Caso genérico create

Create es toda aquella acción que comprende la creación de nuevos registros en la base de datos. Por ejemplo, crear un libro es un caso de uso create. Básicamente, el usuario rellena un formulario; una vez rellenado, se envía una petición al sistema para que guarde los nuevos datos.

En este caso se asume que el control del identificador del nuevo registro creado está controlado por el servidor de base de datos, que es ajeno a Lectums. En general, es un identificador numérico autoincremental.

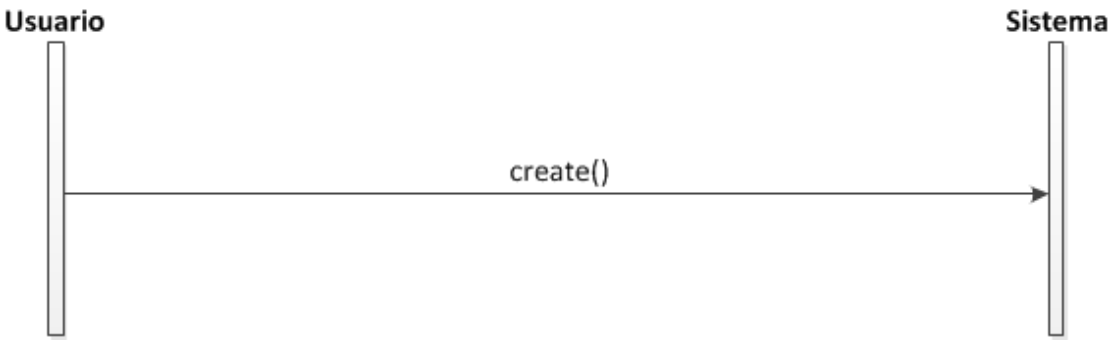

Ilustración 2 Diagrama de secuencia del caso genérico create

Pre

Existe la variable global _POST que contiene los datos del formulario.

Post

Los datos del formulario están guardados.

2.4.2. Caso genérico update

Todas aquellas acciones del usuario en que se actualiza un registro, tales como modificar libro, se basan en el caso de uso genérico update. El usuario accede al registro en cuestión que se le presenta en forma de formulario y, una vez modificados los datos, se hace una petición al sistema para que los guarde actualizados.

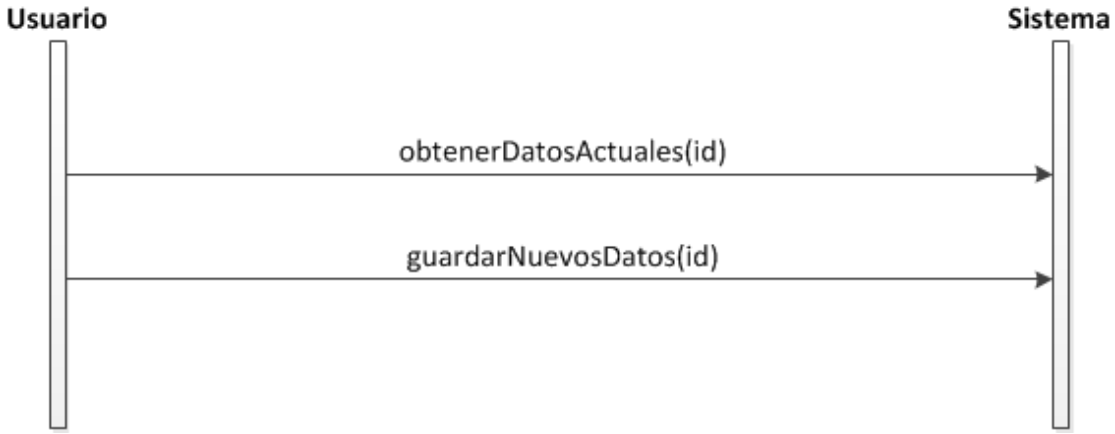

Ilustración 3 Diagrama de secuencia del caso genérico update

obtenerDatosActuales(id)

Pre

Id es un identificador válido

Existe un registro con identificador *id*

Post

El formulario muestra los datos actuales del registro con identificador id

guardarNuevosDatos(id)

Pre

Existe la variable global _POST que contiene los datos del formulario.

Post

El registro con identificador *id* se ha actualizado con los nuevos datos introducidos por el usuario mediante el formulario.

2.4.3. Caso genérico delete

Siempre que el usuario elimina un registro se ejecuta un caso de uso basado en el caso genérico delete. Básicamente el usuario envía una petición al sistema con el identificador del recurso que pretende borrar y el sistema lo elimina.

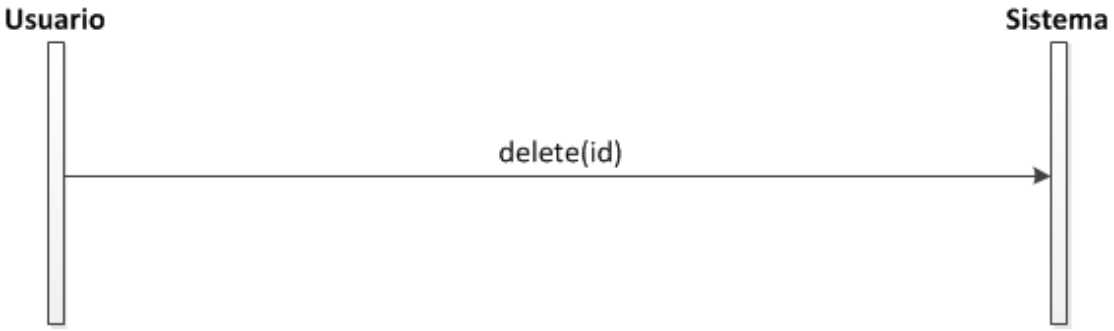

Ilustración 4 Diagrama de secuencia del caso genérico delete

Pre

Id es un identificador válido

Existe un registro con identificador *id*

Post

El registro con identificador *id* no existe.

2.4.4. Caso genérico read

Todos aquellas acciones del usuario que conlleven la recuperación de los datos de un registro concreto se basan en el caso de uso genérico read. *Nombre* sólo se envía como parámetro al sistema por motivos de URL semántica. Se ignora totalmente en la lógica de negocio.

Ilustración 5 Diagrama de secuencia del caso genérico read

Pre

Id es un identificador válido

Existe un registro con identificador *id*

Post

Se muestra al usuario los datos del registro con identificador *id*

2.4.5. Caso finder

Finder hace referencia al buscador que se puede ver en cada página de Lectums y que permite buscador libros, autores, colecciones y editoriales. Es una llamada AJAX, por lo que el funcionamiento no es el habitual de la web.

Ilustración 6 Diagrama de secuencia del caso finder

Pre

Keyword es un identificador válido

Post

Se muestra al usuario un listado de los resultados (si los hay)

2.4.6. Caso me gusta

Me gusta es el enlace que hay en cada libro para que el usuario pueda marcar un libro como su favorito, que luego aparecerá en su sección favoritos. También es una llamada AJAX.

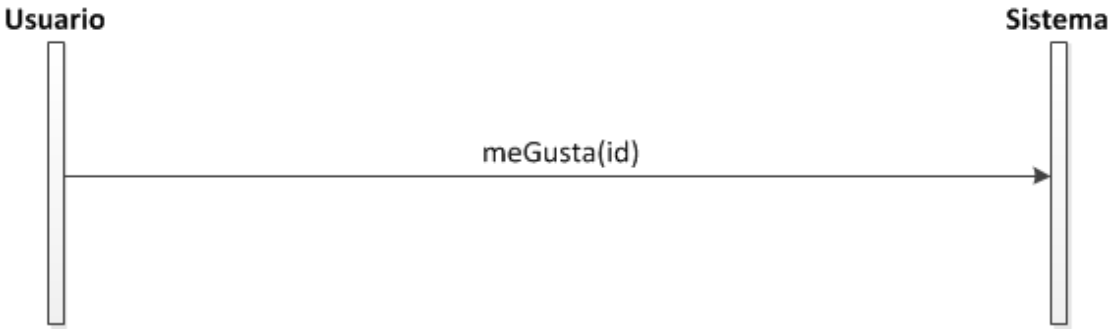

Ilustración 7 Diagrama de secuencia del caso me gusta

Pre

Id es un identificador válido

Existe un libro con identificador *id*

El libro con identificador id no está en la lista de favoritos del usuario

Post

El libro con identificador *id* está en la lista de favoritos del usuario.

2.4.7. Caso no me gusta

Es el caso contrario al anterior, y funciona de la misma manera.

Ilustración 8 Diagrama de secuencia del caso no me gusta

Pre

Id es un identificador válido

Existe un libro con identificador *id*

El libro con identificador id está en la lista de favoritos del usuario

Post

El libro con identificador *id* no está en la lista de favoritos del usuario.

2.4.8. Caso lo tengo

Lo tengo es el enlace que permite al usuario informar al sistema de que tiene un libro en concreto, de forma que éste aparezca en su biblioteca. Funciona bajo AJAX.

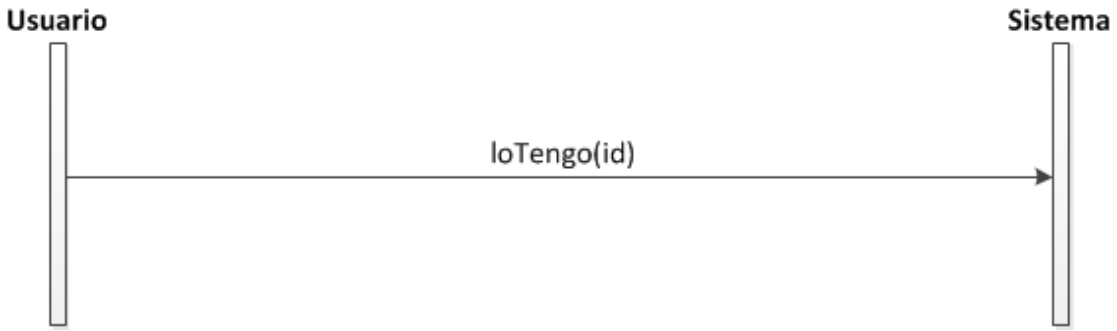

Ilustración 9 Diagrama de secuencia del caso lo tengo

Pre

Id es un identificador válido

Existe un libro con identificador *id*

El libro con identificador id está en la biblioteca del usuario

Post

El libro con identificador *id* no está en la biblioteca del usuario.

2.4.9. Caso no lo tengo

Caso contrario al anterior.

Ilustración 10 Diagrama de secuencia del caso no lo tengo

Pre

Id es un identificador válido

Existe un libro con identificador *id*

El libro con identificador id no está en la biblioteca del usuario

Post

El libro con identificador *id* está en la biblioteca del usuario.

3. Diseño

3.1. Introducción

El diseño de una aplicación de software implica el estudio de la especificación de forma que sea "sencillo" decidir qué funcionalidades son necesarias y de qué forma se deberían implementar, dejando al programador solamente decisiones inherentes al lenguaje de programación. Además, en el caso de una aplicación web, en que la interfaz es tan y tan importante, hay que diseñar esta con cuidado, de forma que en ocasiones las funcionalidades y la forma de implementarlas dependen enormemente de ella.

Anteriormente he defendido que Codeigniter, como cualquier framework MVC, marca claramente la línea a seguir en el diseño de la funcionalidad de cualquier proyecto, en este caso **Lectums**. En esta sección debe quedar más patente, si cabe, ya que marca tanto el diagrama de clases, como el de interacción.

Es obvio que existe una relación muy directa entre el modelo conceptual y los controladores y modelos del sistema MVC. No es momento, bajo mi punto de vista, de hacer un alegato ni en defensa ni contra el modelo tradicional de Ingeniería del Software, pero es evidente que en este proyecto las decisiones de diseño vienen muy marcadas de antemano, y que sólo en casos muy puntuales se requerirá un estudio en profundidad de alguna funcionalidad concreta. Quiero excusar, de esta manera, el no haber explicado aquí métodos del sistema que, por triviales, hubiera resultado absurdo ahondar en ellos. De hecho, no hay condicionantes en este proyecto que impliquen una gran complejidad en ninguno de los casos de uso, lo cual remite necesariamente al diseño. Esto es así en la mayoría de aplicaciones web, en que los denominados métodos CRUD (create, read, update y delete) son mayoritarios; estos métodos, por su condición de triviales, no se explican en su totalidad. Sólo explicaré un ejemplo genérico de cada tipo.

Mención especial a la capa de gestión de datos, que en este caso es una base de datos MySQL de cierta complejidad y que se explicará en el apartado "Diseño de la capa de gestión de datos". Y digo mención especial porque la estructura tiene una complejidad considerable al manejarse en ellas muchos tipos de datos diferentes.

A continuación muestro y explico el Modelo de Componentes, que está compuesto exclusivamente de controladores y modelos, como es obvio en un paradigma MVC. Sobre el diseño y estructura del Framework en sí, hablo en el Apéndice B.

He descrito, adicionalmente, algunos procesos que me han parecido interesantes en forma de modelos de colaboración. Son aquellos casos en que el paradigma MVC no fuerza un modelo a seguir concreto, sino que intervienen las decisiones del analista.

También explicaré todo lo relacionado con el diseño de la interfaz y la usabilidad, haciendo especial hincapié en la metodología que he seguido con el fin de obtener una interfaz usable a la par que vistosa. La simplicidad y el minimalismo han sido la constante.

Muestro, además, algunos detalles acerca de la accesibilidad que he tenido en cuenta a la hora de diseñar la interfaz y la maquetación.

3.2. Modelo de componentes

En el siguiente diagrama se ven todos los controladores y los modelos del sistema; se han excluido las vistas, porque considero que no son, ni deben ser, parte de la lógica, sino que forman parte de la capa de presentación. De hecho una de las reglas más fundamentales de los defensores del desarrollo web estándar es que el contenido y la presentación deben estar siempre separados de la lógica, de forma que alguien sin conocimientos de desarrollo de software pueda modificar fácilmente el aspecto de la salida y que, además, el producto final sea perfectamente modulable. Esto facilita, por ejemplo, el uso de plantillas.

Se excluyen, además, las librerías y helpers (pequeñas funciones que ayudan a hacer tareas concretas) del framework, así como el *core* de este. Estos aspectos se explican en el Apéndice B y, de incluirlos aquí, se haría complicado entender el esquema general del proyecto. Pensemos en ellos como simples *cajas negras* y apliquemos el *principio de ocultación*.

También se excluyen, evidentemente, las variables y estados propios del servidor, como la variable _POST, que tan fundamental resulta a la hora de procesar formularios. Concretamente, _POST es un array global visible por todos los scripts receptores de formularios.

Quiero hacer varias aclaraciones en este punto. El motivo de que sólo se hayan incluido en este modelo de componentes aquellos que he creado yo es que considero que el diseño interno del framework no forma parte de mi proyecto; además, hubiera resultado muy confuso hacerlo a medias, explicando sólo algunas partes, y muy tedioso hacerlo al completo, explicando todo el funcionamiento del framework. Se entiende, además, que el diseño es independiente de la tecnología con lo que, en teoría, el framework Codeigniter no debería formar parte de este diseño. No obstante, tengamos siempre en cuenta que todos los controladores heredan de una clase base *controller* propia de Codeigniter, de la misma forma que los modelos heredan de *model*. Esta herencia permite, entre otras cosas, acceder al core del framework desde cualquier controlador o modelo que creemos. Las visibilidades entre los controladores y los modelos son, en realidad, entre la clase base *controller* y la clase base *model*, aunque para simplificar he preferido colocarlos como si fueran propias de las clases *hijo* que he creado yo. Todas las visibilidades son de tipo atributo; ¿por qué? Al llamar a un modelo desde un controlador o desde otro método se crea un atributo con su nombre accesible, a partir de ese momento, por todos los métodos del

controlador o modelo; esta llamada se hace en el constructor, de forma que es prácticamente automática. De hecho, es perfectamente posible cargar modelos desde el archivo de configuración *autoload*. No existen visibilidades de parámetro, porque nunca se envía un modelo ni mucho menos un controlador por parámetro, sino sólo valores textuales recogidos en las URL o mediante las variables superglobales del sistema. Tampoco hay visibilidades locales, debido al funcionamiento de carga de modelos explicado antes.

Debo decir que en aras de simplificar los diagramas y poder explicar claramente lo que he hecho yo, he obviado algunos detalles del framework que, de ser especialmente estricto, podría haber incluido; no obstante, vuelvo a decir que no creo importante explicar el funcionamiento interno del framework al detalle, sino sólo lo que es intrínseco a mi propio diseño. Para más información, se puede consultar el Apéndice B.

3.2.1. Diagrama de componentes

Hay que recordar que todas visibilidades son de tipo atributo. No se ha representado esto en el modelo para no recargarlo todavía más –hay muchos componentes. Pido disculpas por el número de líneas: en el *listado de componentes* puede verse con más claridad.

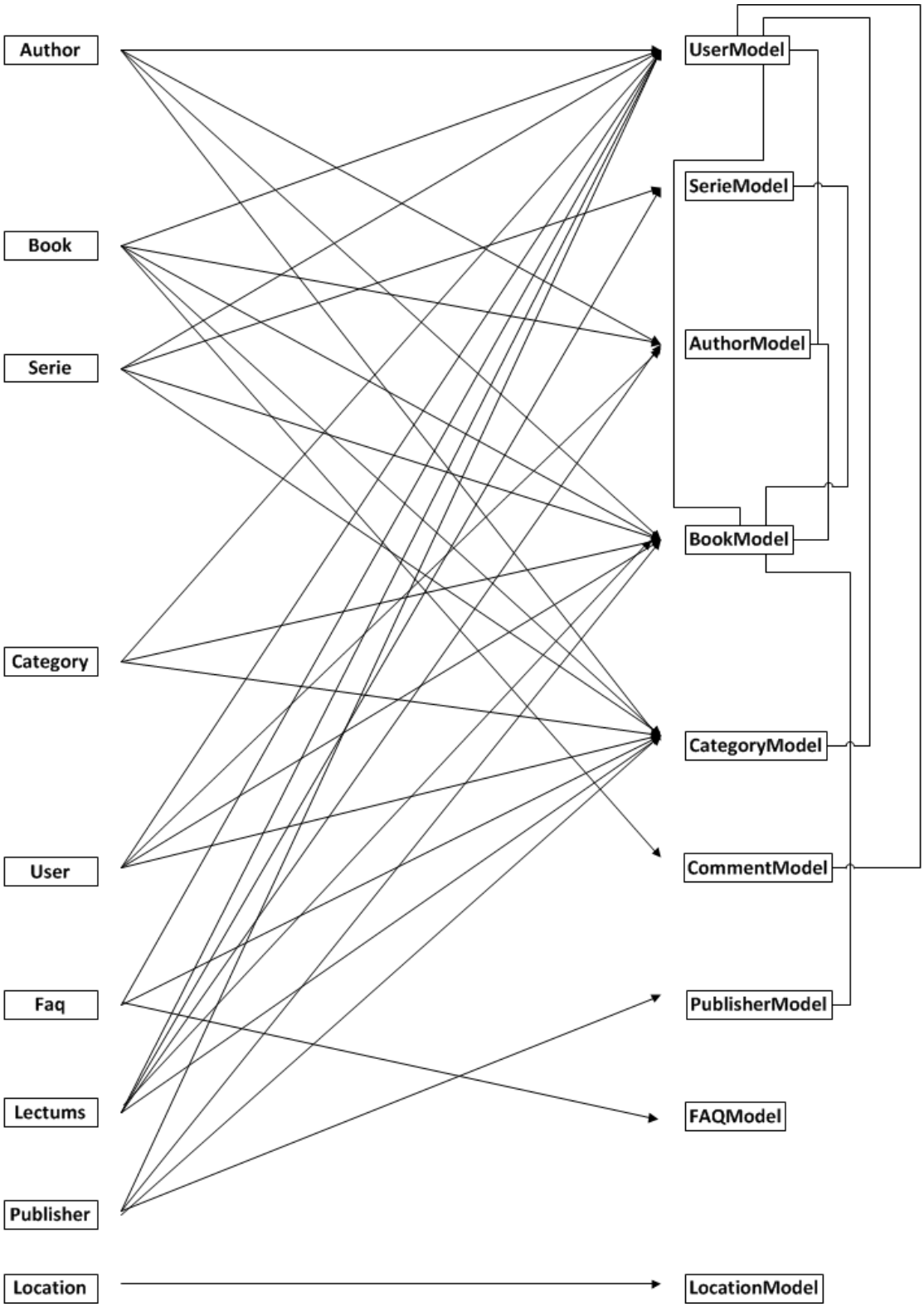

Ilustración 11 Diagrama de componentes

3.2.2. Listado de componentes

Book (Controlador)
Métodos: index(offset, limit), all(offset, limit), read(title, id), finder()
Visibilidades: AuthorModel, UserModel, BookModel, CategoryModel, CommentModel

Author (Controlador)
Métodos: index(offset, limit), all(offset, limit), read(title, id)
Visibilidades: AuthorModel, UserModel, BookModel, CategoryModel

Category (Controlador)
Métodos: read(name, id, offset, limit)
Visibilidades: UserModel, BookModel, CategoryModel

Faq (Controlador)
Métodos: index(), read()
Visibilidades: UserModel, FaqModel, CategoryModel

Lectums (Controlador)
Métodos: index(), finder()
Visibilidades: AuthorModel, UserModel, BookModel, CategoryModel, SerieModel

Location (Controlador)
Métodos: createlocation(), locate(book_id, location_id)
Visibilidades: LocationModel

Publisher (Controlador)
Métodos: index(offset, limit), all(offset, limit), read(title, id), finder()
Visibilidades: PublisherModel, UserModel, BookModel, CategoryModel

Serie (Controlador)
Métodos: read(name, id, offset, limit)
Visibilidades: SerieModel, UserModel, BookModel, CategoryModel

User (Controlador)
Métodos: index(offset, limit), all(offset, limit), read(title, id), login(), logout(), requestpermissions(), posttothewall(message), likebook(id), dislikebook(id), readingbook(id), notreadingbook(id), hasbook(id), nothasbook(id), library(offset, limit), favorites(offset, limit), reading(offset, limit), comment(title, book_id), borrow(book_id), unborrow(book_id), send_message(users_name, users_id), update()
Visibilidades: AuthorModel, UserModel, BookModel, CategoryModel

BookModel (Modelo)
Métodos: getBooksByKeyword(keyword, limit), numBooks(), getTitlesByUserReading(id), getBooks(offset, limit), numBooksByUserID(id), getBooksByUserID(id, offset, limit), numFavoritesBooksByUserID(id), numReadingBooksByUserID(id), getFavoritesBooksByUserID(id, offset, limit), getReadingBooksByUserID(id, offset, limit), getLastestsBooks(limit), getBookByID(id), getBooksByAuthorID(id), getBooksByPublisherID(id), getBooksByCategoryID(id, offset, limit), numBooksByCategoryID(id), numBooksBySerieID(id)
Visibilidades: AuthorModel, PublisherModel, SerieModel

AuthorModel (Modelo)
Métodos: getAuthors(offset, limit), numAuthors(),getAuthorByID(id), getAuthorsByBookID(id), getAuthorsByKeyword(keyword, limit)
Visibilidades:

CategoriesModel (Modelo)
Métodos: getCategories(),getCategoryNameByCategoryID(id)
Visibilidades:

CommentsModel (Modelo)
Métodos: getCommentsByBookID(id)
Visibilidades:

FaqModel (Modelo)
Métodos: getFaqs(), numFaqs(),getFaqByID(id)
Visibilidades:

LocationModel (Modelo)
Métodos: createLocation(name), getLocations(), locate(book_id, location_id)
Visibilidades:

PublisherModel (Modelo)
Métodos: getPublishers(offset, limit), numPublishers(), getPublisherByID(id), getPublishersByBookID(id), getPublishersByKeyword(keyword, limit)
Visibilidades:

SerieModel (Modelo)
Métodos: getSeriesByBookID(id), getSeriesByKeyword(keyword, limit), getSerieNameBySerieID(id)
Visibilidades:

UserModel (Modelo)
Métodos: standardLogin(data), logout(), getUsers(offset, limit), update(data), sendMessage(users_id, data), getUserByID(id), numUsers(), logged(), getName(), likebook(id), dislikebook(id), readingbook(id), notreadingbook(id), hasbook(id), nothasbook(id), borrow(book_id, name), unborrow(book_id), getMessagesByUserID(id), comment(user_id, book_id, message)
Visibilidades: AuthorModel, BookModel, CategoryModel, CommentModel

3.3. Diagramas de colaboración

En este punto, quiero exponer algunos de los diagramas de colaboración que considero básicos, entre ellos un ejemplo de cada tipo CRUD. Como ya he dicho antes, no voy a incluir elementos tales como librerías o ayudantes del framework, ni aquellos elementos propios del lenguaje. En el Apéndice B hay una pequeña explicación del funcionamiento interno del framework.

3.3.1. Caso genérico: create

Create se refiere a la acción de introducir nuevos registros en la base de datos. Conceptualmente es aplicable, por ejemplo, a la acción de crear un nuevo libro. En este caso de uso genérico intervienen por un lado el usuario, que introduce datos en un formulario, y por el otro el controlador y el modelo que representan el concepto sobre el que se va a crear; puede haber más de un modelo implicado si se van a introducir datos en varias tablas, aunque no es muy frecuente.

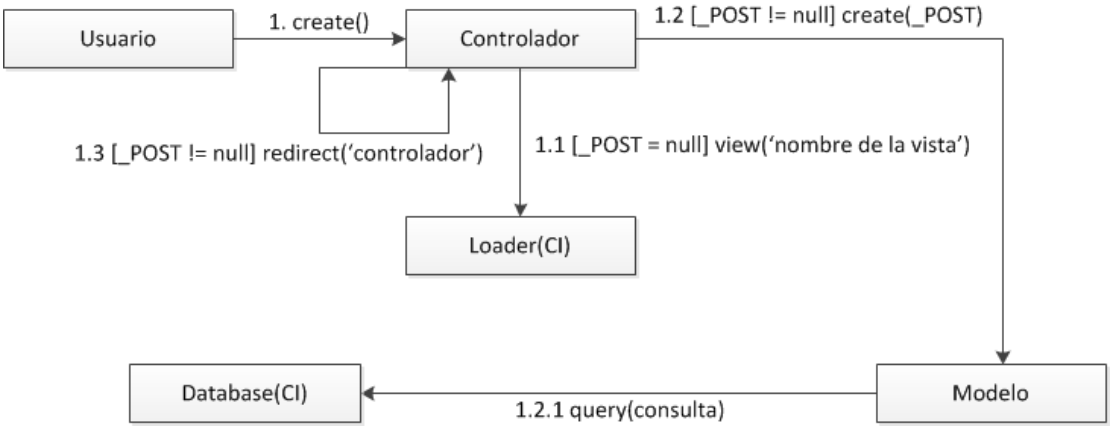

Ilustración 12 Diagrama de colaboración del caso genérico create

Hay una variable a tener en cuenta, que es *_POST*; *_POST* es una variable global que contiene la información enviada por el usuario mediante el formulario HTML. ¿Quién recibe el formulario? El controlador, de forma que envía esa variable _POST por parámetro al modelo –en este caso la he llamado datos para ilustrar- y este, a su vez, realiza la consulta pertinente a la base de datos, en este caso de creación. Como es un agregación de registro en la base de datos, el modelo no retorna nada al controlador. Una vez el controlador ha hecho la solicitud al modelo, carga la vista

correspondiente que, en este caso, podría ser un mensaje confirmando que se ha insertado contenido nuevo en la base de datos.

Evidentemente *consulta* es una consulta SQL a la tabla correspondiente al modelo. La función query, de la clase db, accesible desde cualquier modelo, proporciona cierta protección contra SQL injection.

3.3.2. Caso genérico: read

Read se refiere a la acción de recuperar datos de la base de datos. Pueden ser tanto listas de registros como un registro particular, como registros de varias tablas a la vez. En este caso interviene el usuario, que hace una petición a una URL concreta, ya sea directamente o mediante un enlace, y el controlador y el modelo aplicables a ese concepto; de hecho, es perfectamente posible que varios modelos intervengan si la consulta está relacionada con varias tablas: es el caso, por ejemplo, de un libro, que además recupera información del autor.

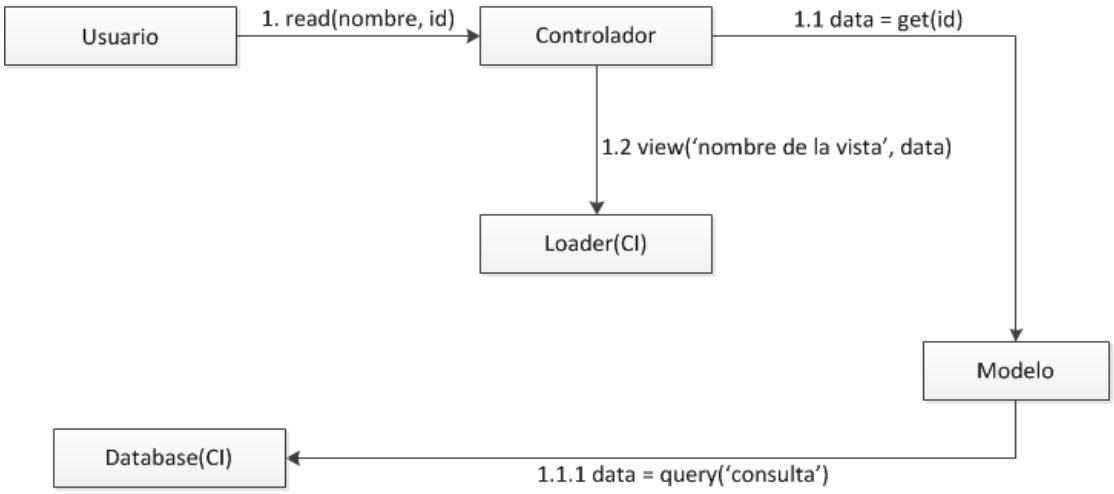

Ilustración 13 Diagrama de secuencia del caso genérico read

Aquí lo importante es ver cómo el controlador solicita los datos pertinentes al modelo y una vez obtenidos –cuando el modelo retorna los datos de la base de datos- el controlador carga la vista pasándole la información obtenida. Hemos de entender el controlador como un simple intermediario; idealmente debería haber poca lógica en el controlador. En este caso genérico no he especificado qué quiero leer para clarificar el ejemplo, pero se podría pasar por parámetro al controlador la id del registro que queremos obtener, por ejemplo la id de un libro. Además, en la mayoría de los casos, paso también el nombre del elemento; no se utiliza en la lógica, sólo sirve para mostrarlo por URL y agregar semántica.

3.3.3. Caso genérico: update

Update se refiere a la acción de modificar registros en la base de datos. Conceptualmente es aplicable, por ejemplo, a la acción de modificar un determinado libro. En este caso de uso genérico intervienen por un lado el usuario, que modifica datos de un formulario, y por el otro el controlador y el modelo que representan el concepto sobre el que se va a modificar; es posible que intervenga más de un modelo, análogamente a lo que ocurre con el caso genérico *create*.

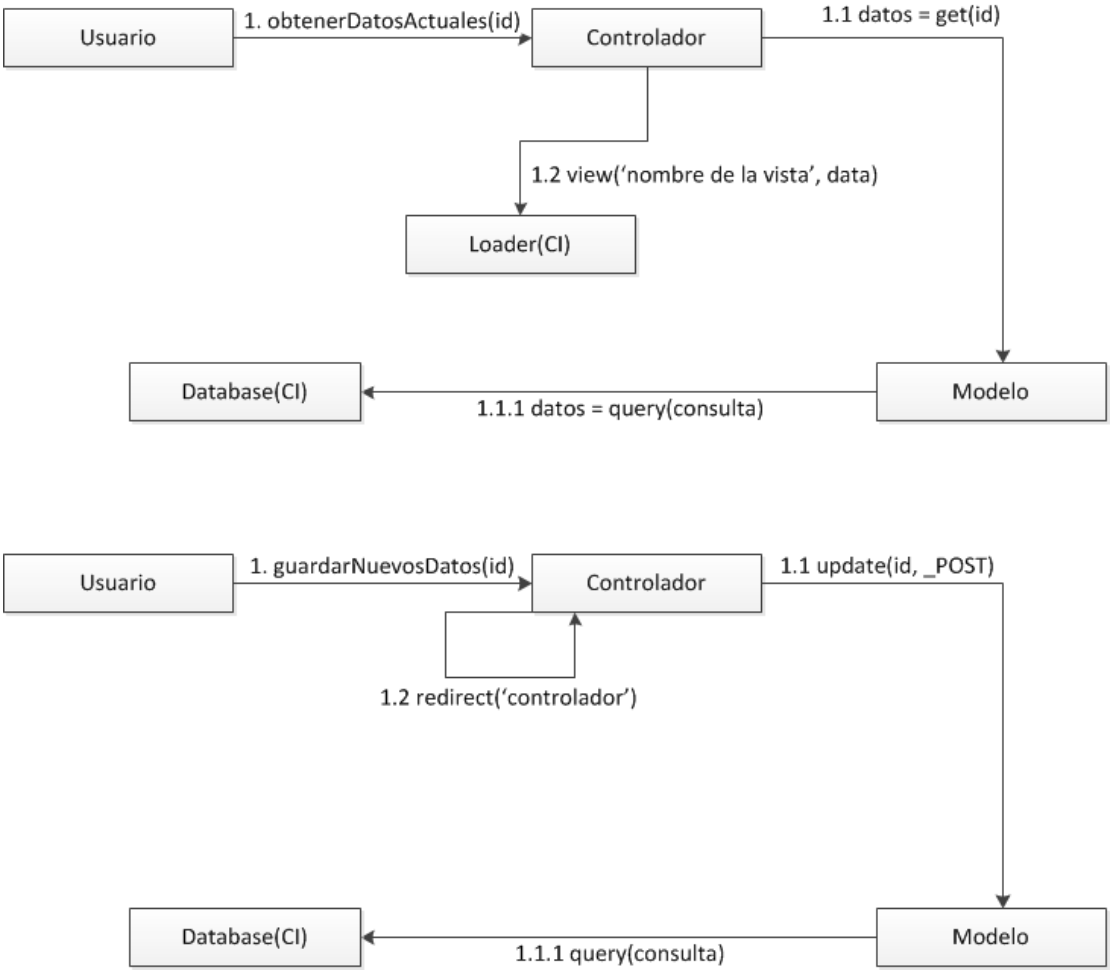

Ilustración 14 Diagrama de colaboración del caso genérico update

Este caso es muy similar al caso de creación, de hecho la principal diferencia es que en este caso el usuario no va a ver un formulario vacío, sino uno con los datos anteriores del registro que está modificando. Al enviar no crea un registro nuevo, sino que lo actualiza con los nuevos datos.

3.3.4. Caso genérico: delete

Delete se refiere a la acción de borrar registros de la base de datos, por ejemplo borrar un libro. Suele ser el caso más sencillo, aunque a veces tiene implicaciones en varias tablas; si, por ejemplo, borramos un autor, ¿deberíamos borrar sus libros? Interviene el usuario, que ejecuta una

URL ya sea directamente o mediante un enlace, y el controlador y el modelo o modelos que representan el concepto a borrar.

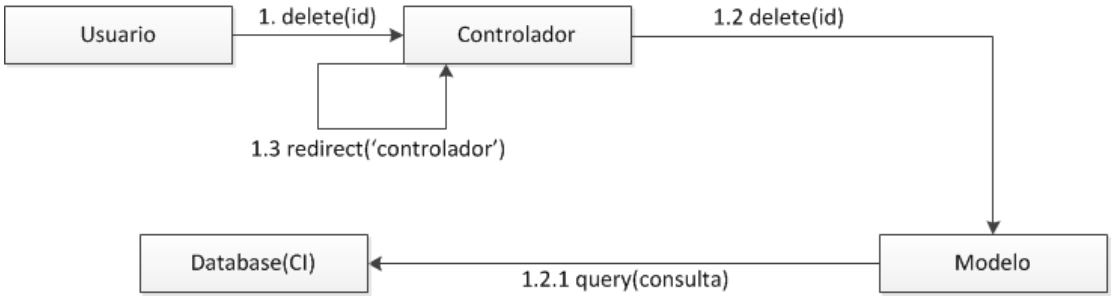

Ilustración 15 Diagrama de colaboración del caso genérico delete

Borrar es probablemente el caso más sencillo; simplemente se hace una petición al controlador en la que se especifica el identificador –el nombre aparece para dar semántica a la URL- y el controlador solicita al modelo que borre el registro. Hay casos en que hay que hacer borrados en cascada, por lo que el modelo puede llegar a solicitar datos y a hacer peticiones a otros modelos. Nunca un controlador puede llamar a otro controlador –debido a su estrecha relación con las URL's, se produciría una redirección.

3.3.5. Caso: finder

Los casos genéricos anteriores representan no sólo la mayoría de métodos existentes en Lectums, sino la mayoría de métodos existentes en cualquier proyecto web. No obstante, quiero destacar también el método *finder*, que da vida al buscador existente en todas las páginas. Se trata de una modificación del caso *read*, por supuesto, pero vale la pena explicar su funcionamiento.

En el caso *finder* hay diferencias notables en cuanto a la interacción del usuario; éste no ejecuta una URL, sino que hace una petición dinámica mediante AJAX a un método de un controlador concreto. Hablamos del controlador Lectums, que me sirve para aunar algunos casos de uso que no tienen un concepto particular al que estar asociados.

Hay que explicar, primero de todo, qué es AJAX. AJAX son las siglas de *Asynchronous JavaScript and XML*, que no es más que una manera un tanto rimbombante de definir una solución a un problema histórico de la web: su falta de dinamismo y de estado. Sin AJAX, cualquier petición al servidor requiere un cambio de página, como es patente cuando, por ejemplo, clicamos en un enlace. Con AJAX esto ya no es necesario; mediante JavaScript hacemos una petición directa al servidor que nos responde, también mediante JavaScript, con algún tipo de dato, que suele ser XML –aunque también puede ser JSON o texto plano. También mediante

JavaScript procesamos dicha respuesta y modificamos la vista con código del lado del cliente, sin cambiar de página ni hacer más peticiones al servidor.

Se puede comprobar el efecto cuando, al introducir una palabra en el buscador de Lectums, nos aparecen resultados en una lista desplegable sin haber hecho ningún cambio de página. No sólo aporta dinamismo sino que es una manera muy *usable* de proporcionar resultados de búsqueda – Google intenta explotar esto con *Google Instant*.

¿Y cómo podemos esquematizar esto? De la siguiente manera:

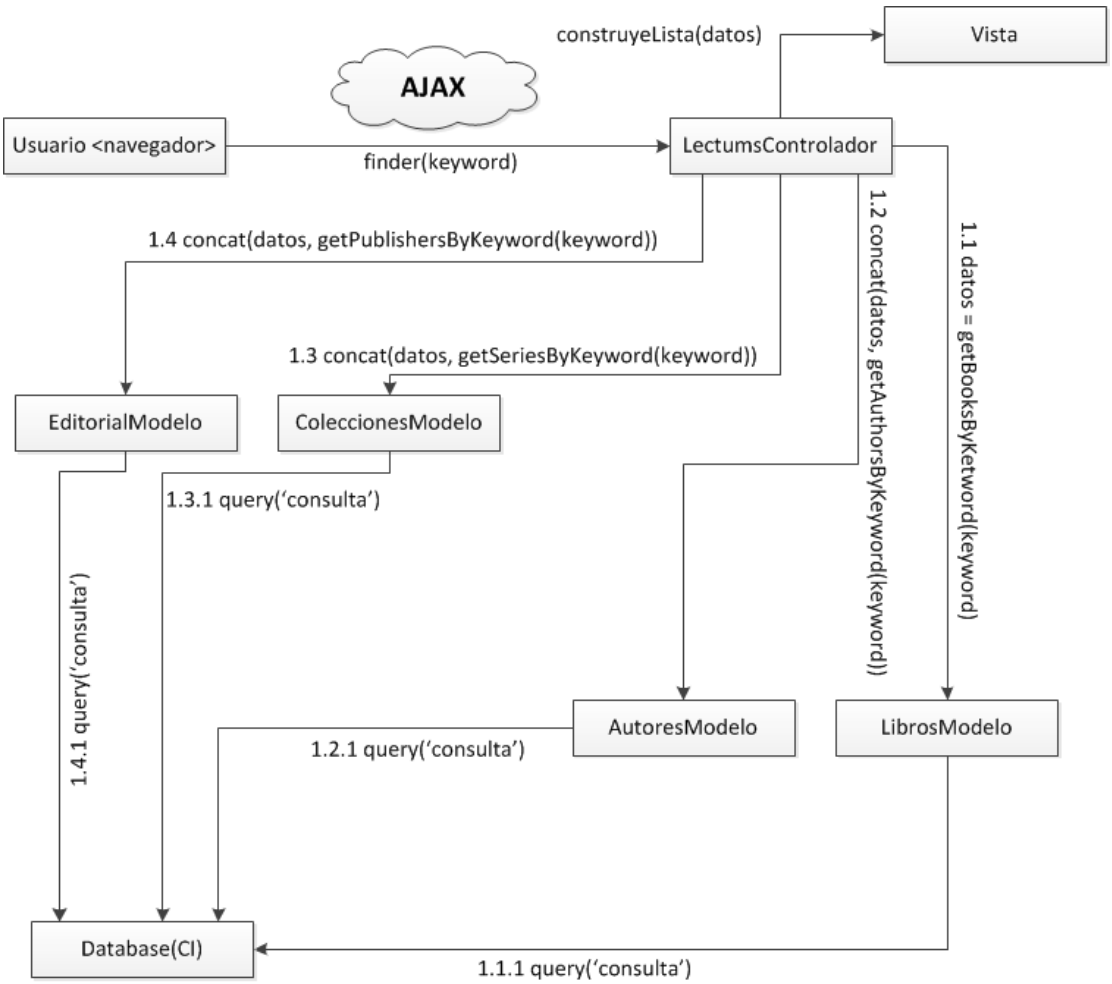

Ilustración 16 Diagrama de colaboración del caso finder

Aunque el esquema parezca complejo en realidad el funcionamiento no lo es tanto. Primeramente hay que notar la *nube AJAX*. Me he permitido la licencia de utilizar una nube para representar que tanto la petición que hace el usuario al controlador como el envío de información que hace el controlador al usuario –al navegador, en realidad- no son peticiones normales, sino que funcionan a través de AJAX como he explicado antes. Una vez comprendido esto, lo demás es fácil de

entender: el controlador, al recibir la petición, le pide a los 4 modelos que contienen datos a mostrar en el buscador que envíen los ítems que concuerdan con la palabra clave *keyword*; cuando tiene los 4 listados, los concatena y los devuelve al navegador. Una vez allí, JavaScript –JQuery realmente- se encarga de generar la lista.

3.3.6. Caso me gusta y no me gusta

Me gusta y *no me gusta* son llamadas AJAX que se producen cuando el usuario clica en los botones "me gusta" y "no me gusta". Esa llamada actúa sobre el controlador User (Controlador) que, a su vez, solicita al modelo la acción indicada; el modelo (UserModel) envía una consulta a la base de datos que actualiza las tablas correspondientes. Esta acción desencadena en el lado del cliente que JQuery cambie el botón "me gusta" por "no me gusta" y viceversa.

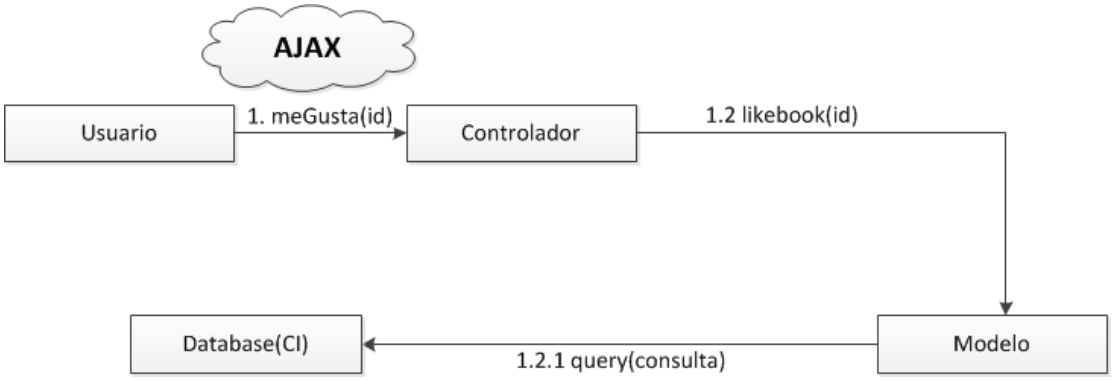

Ilustración 17 Diagrama de colaboración del caso me gusta

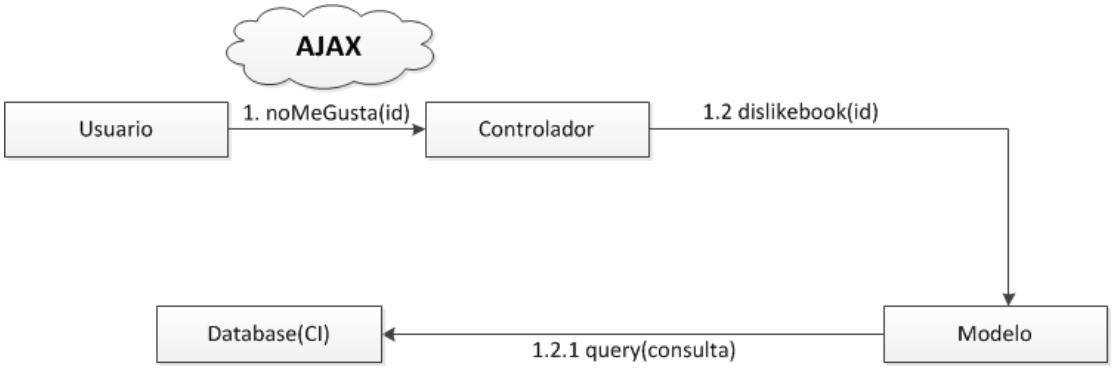

Ilustración 18 Diagrama de colaboración del caso no me gusta

3.3.7. Caso lo tengo y no lo tengo

Lo tengo y *no lo tengo* son llamadas AJAX que se producen cuando el usuario clica en los botones "lo tengo" y "no lo tengo". Esa llamada actúa sobre el controlador User (Controlador)

que, a su vez, solicita al modelo la acción indicada; el modelo (UserModel) envía una consulta a la base de datos que actualiza las tablas correspondientes. Esta acción desencadena en el lado del cliente que JQuery cambie el botón “lo tengo” por “no lo tengo” y viceversa.

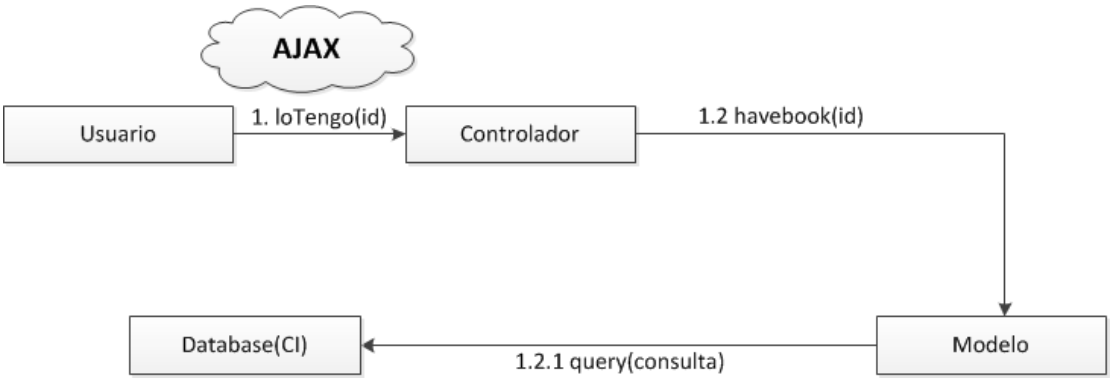

Ilustración 19 Diagrama de colaboración del caso lo tengo

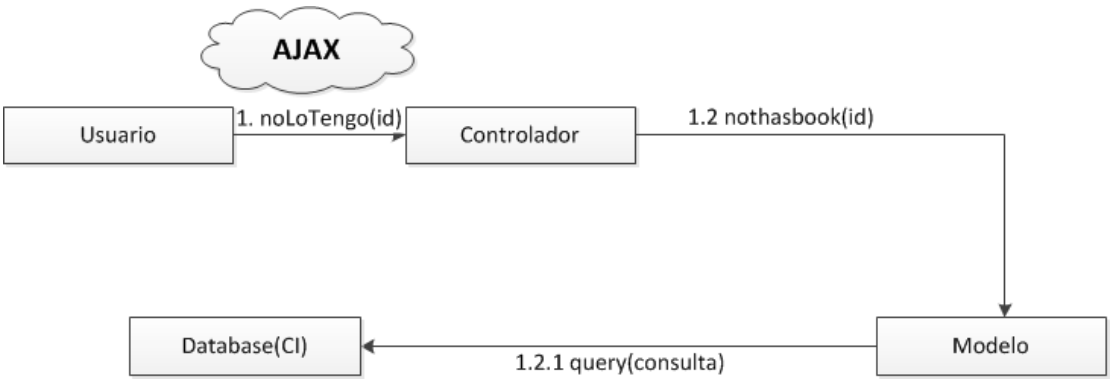

Ilustración 20 Diagrama de colaboración del caso no lo tengo

3.4. Diseño de la capa de gestión de datos

La capa de gestión de datos, en este caso, es una base de datos MySQL; cabe destacar la complejidad de su esquema, consistente en mas de veinte tablas, muchas de ellas representando relaciones muchos a muchos.

La forma de obtener este esquema ha sido, precisamente, proyectar el modelo conceptual a las necesidades de datos del proyecto. De esta forma, la gran mayoría de entidades se han convertido en tablas; las relaciones presentes entre las entidades han desembocado, algunas de ellas, en nuevas tablas que hacen de puente para relaciones muchos a muchos.

Esta complejidad ha provocado que los modelos, que son los encargados de recoger esos datos a la medida de las necesidades, tengan, en algunos casos, complejos métodos que hacen consultas compuestas y, en algunos casos como por ejemplo el de las categorías, multinivel. De hecho, creo que vale la pena destacar el funcionamiento de la categorías, que representan un árbol genérico en el que todas las ramas –no sólo las hojas- pueden tener libros asociados. No deja de ser una tabla con una relación reflexiva que utiliza la clave *parent* para identificar la relación.

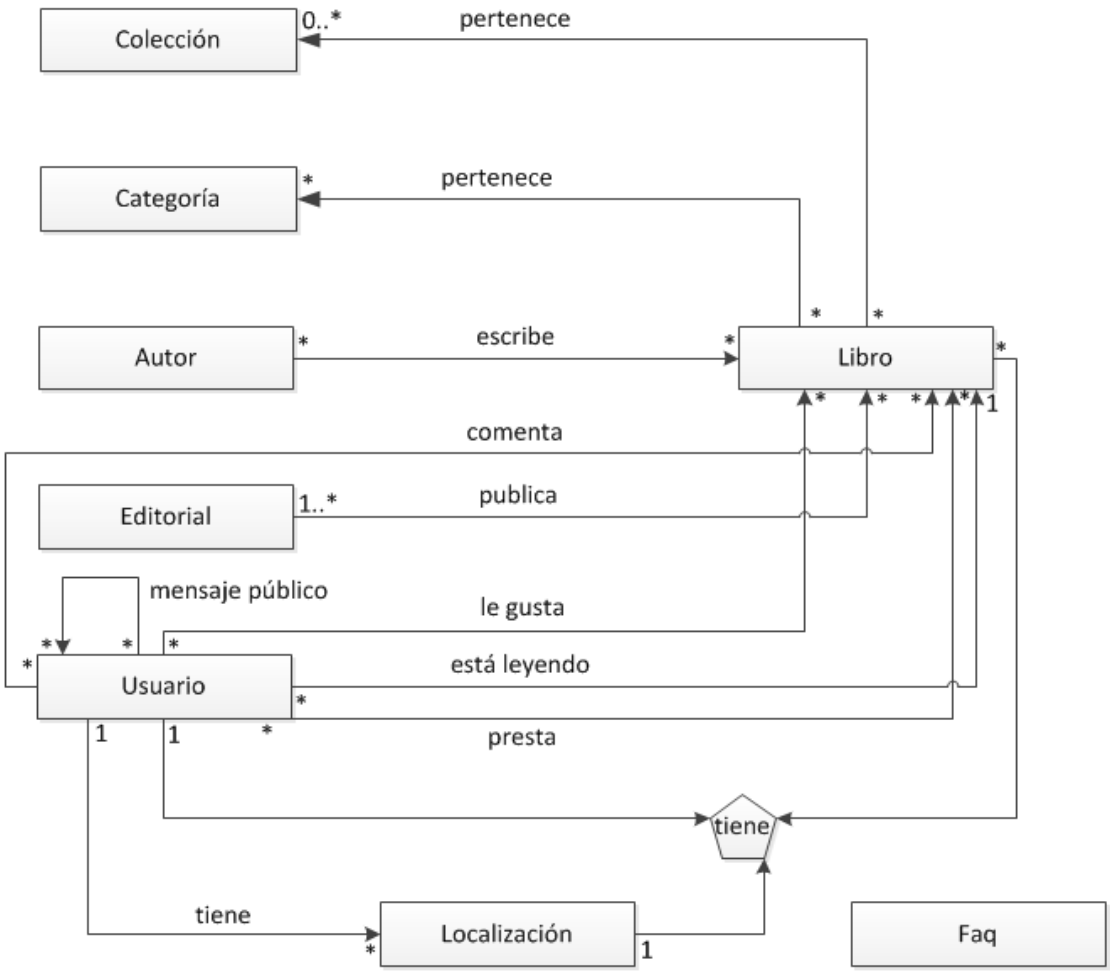

Ilustración 21 Diagrama ER o de Chen de la base de datos

Como se puede apreciar, este esquema es prácticamente un calco del modelo conceptual. ¿Por qué? Muy sencillo, porque con el paradigma MVC existe una relación intrínseca entre modelos –que recuerdo eran casi todas las entidades del modelo conceptual- y tablas de la base de datos.

3.5. Diseño de interfaz

3.5.1. Introducción

Mi experiencia en cuanto a diseño para la web se basa en los años de trabajo como programador en una empresa de desarrollo web al lado de dos diseñadores. Aunque probablemente el propósito

de una Ingeniería no sea el de diseñar, ni a nivel gráfico ni a nivel de usabilidad, sí que creo que el esfuerzo puesto en este proyecto en ambos campos tiene una importancia capital y creo necesario explicar algunas de las claves del desarrollo.

En cuanto a la accesibilidad, he seguido las pautas del W3C en referencia a la adaptación de páginas web para personas con algún tipo de impedimento físico, intelectual o, simplemente, a personas con equipos de hardware limitados o especiales.

3.5.2. Diseño gráfico

Gráficamente he querido darle un toque minimalista y sencillo, con colores pastel de muy fácil "digestión". Al ser una página web dedicada en cierta manera al contenido, la legibilidad de los textos es una parte importante.

Siguiendo las lecciones que se pueden extraer de aplicaciones web como Facebook o Tuenti, que además son redes sociales, los colores se limitan a 3 o 4 y se sigue una coherencia total en todo el diseño.

Para los prototipos utilicé un simple papel, y luego, una vez decidida la estructura, utilicé Photoshop para definir el aspecto que debía tener la aplicación.

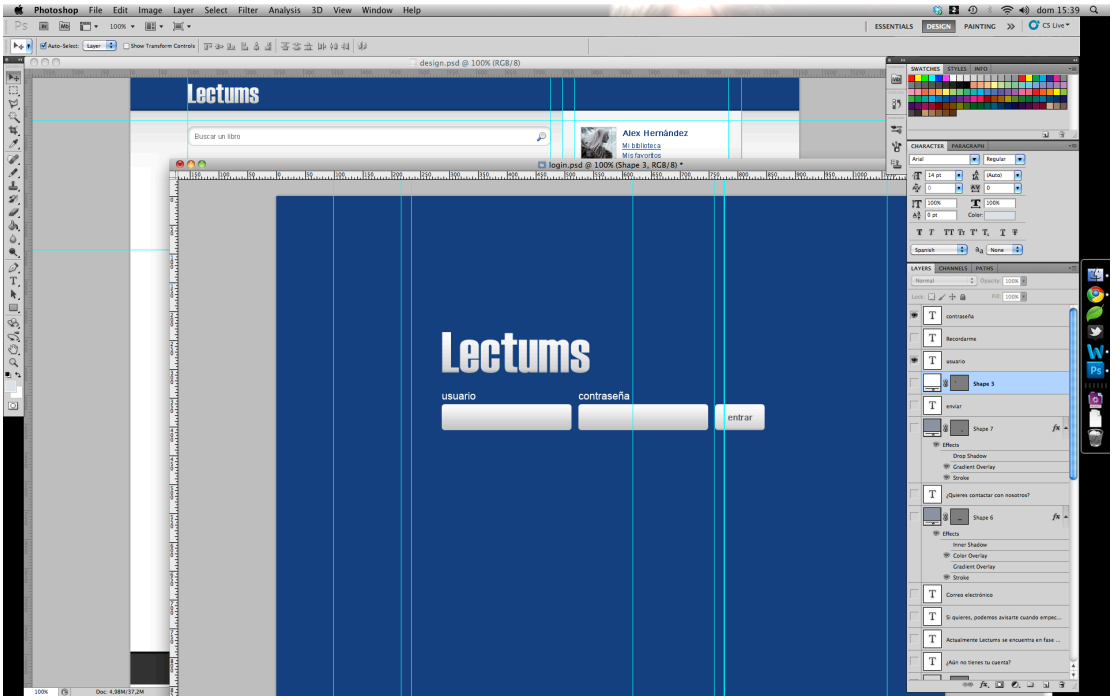

Ilustración 22 Diseños originales en Photoshop

Un aspecto apreciable es la fidelidad con que he seguido –y se ha de seguir- el diseño en Photoshop a la hora de realizar la maquetación. En principio, un diseñador gráfico que desarrolla

páginas web no debería dejar margen de elección al maquetador en cuanto a la apariencia o la usabilidad del sitio web, pues es tarea del diseñador gráfico lo primero y lo segundo, sino de él, de un experto en usabilidad, no del maquetador, que tiene un perfil técnico. Como en mi caso yo he hecho de todos ellos, es difícil marcar dónde empieza una cosa y dónde acaba otra, pero sí que he intentado ser fiel al diseño original.

3.5.3. Usabilidad

La usabilidad es la capacidad de una interfaz para ser entendida y utilizada por los usuarios. Esto es fundamental en una aplicación de una cierta complejidad como **Lectums**, lo que hace que haya invertido un esfuerzo considerable en ello. Hay infinidad de páginas web en las que explican las claves de una buena usabilidad, y no es el propósito de esta memoria hacer un estudio detallado de cada una de ellas, pero sí quisiera destacar las que creo más importantes entre las que he tenido en cuenta a la hora de desarrollar **Lectums**:

- Regla del 5 y del 3: nunca debería haber más de 5 opciones de menú, y nunca debería haber más de 3 niveles para cada una de ellas. Esta es una regla perfectamente cumplida por **Lectums** excepto en el caso de las categorías, que obviamente son más de 5. No obstante, no se puede considerar la lista de categorías un menú al uso y, en realidad, no es imprescindible.

- Una herramienta de búsqueda eficaz: es muy sencillo introducir un libro o un autor en el buscador de **Lectums** y clicar en los resultados de la lista.

Ilustración 23 Finder

- Espacio blanco: esta es una regla tanto de diseño como de usabilidad. En toda aplicación o página web debe existir espacio blanco suficiente como para que no sea recargada y despiste.

- *Call to actions*: *call to action* es un concepto utilizado para definir aquellos botones, enlaces, banners… en definitiva, cualquier tipo de elemento que llama poderosamente la atención e invita al usuario a realizar una determinada acción. Si dejamos que sólo ciertos elementos importantes destaquen, entonces conseguimos que esos elementos tengan un ratio de acceso muy superior. En **Lectums** he utilizado esta técnica para las opciones más importantes, como el propio buscador o las opciones superiores “Libros” y “Autores”.

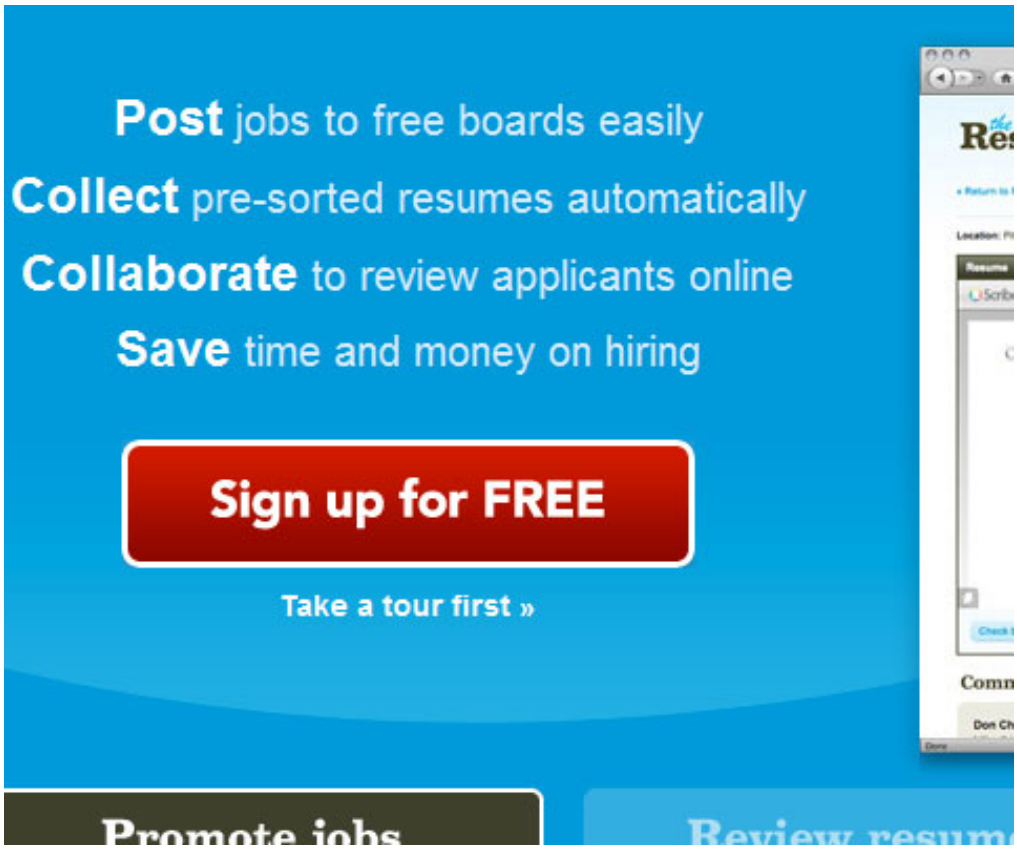

Ilustración 24 Ejemplo de Call to Action

- Interfaz homogénea: qué cada página de la aplicación tenga una interfaz totalmente diferente puede resultar atractivo, pero no es usable; el usuario pierde la capacidad de recordar y aprender, pues cada vez tiene una barrera de entrada. Por lo tanto, en **Lectums** he intentado que la interfaz sea bastante homogénea, dejando las cosas diferentes para los *call to actions* o los puntos en que verdaderamente era necesario.

- No utilizar *frames*: esto es evidente, es una muy mala práctica que tiene su origen en una necesidad insatisfecha, poder reutilizar diferentes partes de la página web. Esto se consigue en la actualidad con "includes" del lenguaje en el lado del servidor, y los *frames* ya no sólo no son necesarios sino que además son negativos para la usabilidad. **Lectums** no usa frames, excepto en un apartado concreto del admin, para simular un efecto imposible de hacer de otra manera –carga dinámica de imágenes.

- Probar la web en distintos navegadores y asegurarse de que tiene el mismo funcionamiento en todos, en la medida de lo posible. En **Lectums** esto es así, excepto en algunos redondeados que con toda seguridad no funcionarán en versiones antiguas de Internet Explorer, aunque no son imprescindibles.

- Color homogéneo para los enlaces y subrayado: no siempre queda bonito subrayar los enlaces, pero es una ayuda visual a la hora de identificarlos que los usuarios agradecen. En **Lectums** casi todos los enlaces están subrayados y tienen el mismo color.

3.5.4. Accesibilidad

La accesibilidad en el software consiste en adaptar en la medida de lo posible las aplicaciones para que personas con problemas físicos, psicológicos o de cualquier otra consideración puedan acceder en igualdad de condiciones. Como en el apartado de usabilidad, no es mi intención explicar todas y cada una de las reglas que rigen esta materia, pero si algunas de las más importantes que se han seguido al desarrollar **Lectums**:

- Maquetación siguiendo los estándares: durante mucho tiempo se maquetó con tablas, elementos anticuados de HTML y etiquetas inadecuadas. Hay que maquetar siguiendo los estándares del W3C, en concreto limitando el uso de tablas al contenido tabular, no a la estructuración de la página web. En **Lectums** no se utilizan tablas.

- Utilizar los atributos *title* y *alt* en los elementos correspondientes, para añadir información semántica a elementos que no la tienen, como imágenes.

- Evitar el uso de Flash y de imágenes cuando pueden haber textos. Ni los elementos Flash ni las imágenes pueden leerse con navegadores para ciegos.

- Colores con suficiente contraste para personas que, sin ser ciegas, tienen problemas de visión.

- Fuentes y tamaños de letra suficientemente grandes para que personas con deficiencias visuales no tengan problemas para leer el texto; incluso sin tener ningún problema de visión, las letras grandes ayudan a leer textos largos. No obstante, el tamaño de la letra suele ser inversamente proporcional a la belleza visual, por lo que hay que buscar un equilibrio.

Hay otras muchas reglas de accesibilidad. De hecho, esta materia es muy controvertida porque, a veces, los defensores más radicales de la accesibilidad llegan a exigir técnicas y normas tan rígidas que afectan muy negativamente a la experiencia de los usuarios que no tienen problemas que, de hecho, son la inmensa mayoría. Por tanto es muy importante buscar un equilibrio racional de forma que, al mismo tiempo que garantizas una experiencia agradable para la mayoría de usuarios, no crees barreras de entrada insalvables en los que tienen algún problema de acceso.

Un aspecto pocas veces tenido en cuenta es que no todos son problemas físicos o psicológicos; a veces sencillamente el usuario dispone de un hardware muy antiguo o muy moderno, como los

dispositivos móviles. En estos casos normalmente es suficiente con hacer una maquetación correcta.

4. Implementación y tests

4.1. Introducción

Es importante elaborar una estrategia tanto de desarrollo como de testeo para poder gestionar un proyecto de esta magnitud. Existen multitud de métodos de desarrollo de software, desde el tradicional, que en cierta manera es el que defiende la Ingeniería de Software clásica, hasta métodos más novedosos –ya no tanto- basados en varios ciclos mucho más cortos de desarrollo, como Scrum o Extreme Programming, llamados también métodos ágiles.

4.2. Método clásico

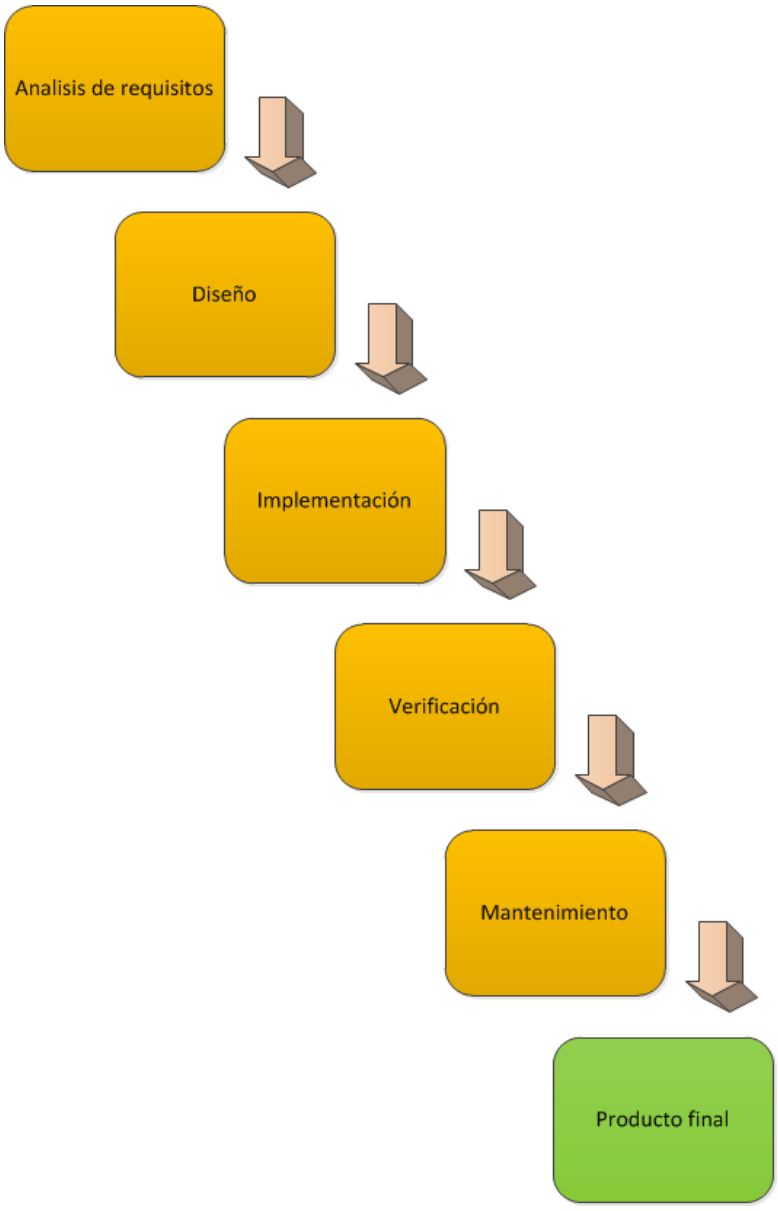

Ilustración 25 Modelo clásico o de Waterfall

En esencia, el método clásico estipula un periodo largo de especificación y diseño del software en su conjunto, para después pasar a implementarlo siguiendo la documentación generada. En principio, no se han de tomar decisiones en el periodo de implementación más allá de aspectos concretos del lenguaje, puesto que se supone que todas las decisiones de diseño se han tomado ya. Por supuesto, podemos hablar de ventajas y desventajas:

A continuación algunas de las ventajas del método clásico:

- Permite abordar proyectos de software grandes *de una pieza*. Esto significa que es fácil mantener la coherencia porque todo el software se ha diseñado a la vez, pensando en su conjunto.

- La implementación es trivial, puesto que todas las decisiones importantes se han tomado de antemano. Debido a esto, es relativamente sencillo dividir todo el software entre varios desarrolladores, pues que la documentación especifica claramente cada parte.

- Es el método clásico y, por tanto, la mayoría de desarrolladores, incluidos los universitarios, lo dominan.

Algunas desventajas:

- No ofrece margen de maniobra: en mercados cambiantes en que un producto de software puede carecer de sentido con el paso de unos pocos meses, o en el que la aparición de un competidor puede hacer que nuestro software deba cambiar rápido, el método clásico es muy poco flexible, puesto que todo es un conjunto interrelacionado.

- Ofrece muy poco feedback al cliente, que sólo empezará a ver el producto cuando prácticamente está acabado. Esto hace que en algunos casos proyectos terminados se pierdan.

- Exceso de documentación: se generan cantidades ingentes de documentación que en muchos casos no son sino un escollo.

- Exceso de reuniones o *meetings*: manejar un proyecto grande al completo requiere de muchas reuniones muy largas que en muchos casos son absolutamente improductivas.

- Limita la creatividad de los desarrolladores, que encuentran poca capacidad de cambio en especificaciones tan rígidas.

4.3. Métodos ágiles

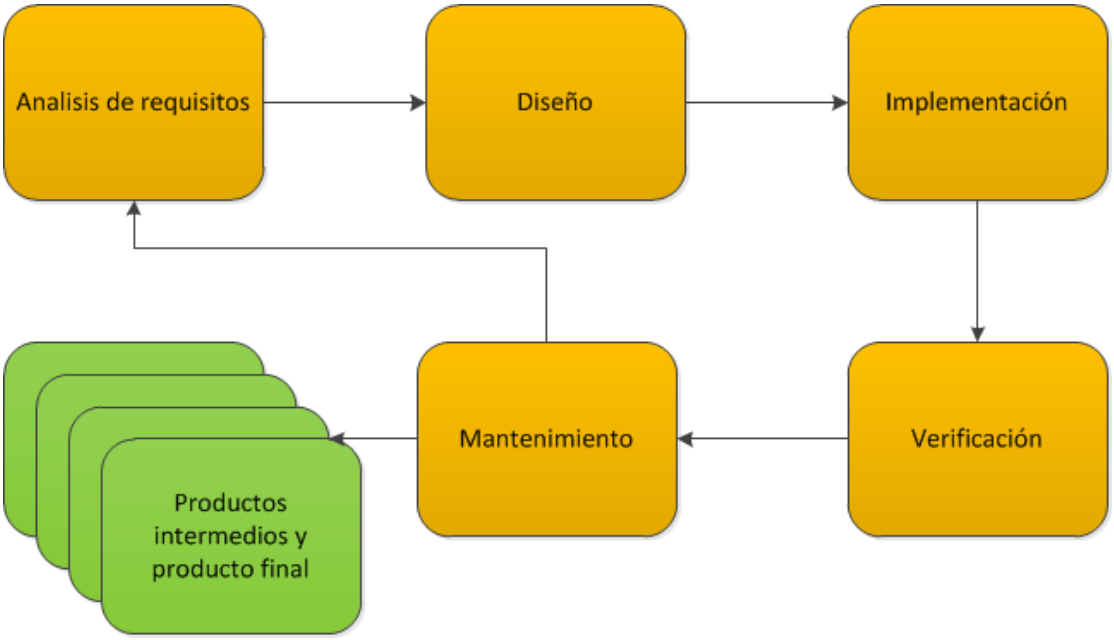

Ilustración 26 Metodología ágil

Los métodos ágiles se basan en desarrollar el software por partes; básicamente, se divide el software en pequeñas subaplicaciones que desde el primer momento son usables, aunque obviamente con menos funcionalidades que el software final. Cada subaplicación es un ciclo en el que se realiza todo el proceso de desarrollo de software. Como el método clásico, esto tiene ventajas e inconvenientes:

Algunas ventajas de los métodos ágiles:

- Cada ciclo produce una pequeña aplicación: en caso de cancelación del proyecto, no tenemos un montón de líneas de código que no hacen nada, sino una pequeña aplicación, o varias, que hacen pocas cosas, pero que las hacen bien.

- Limita la especificación y prácticamente carece de diseño: al estar trabajando con aplicaciones pequeñas, una documentación extensa es prácticamente innecesaria; el peso del desarrollo descansa en los propios desarrolladores, que tienen mucho margen de decisión.

- Limita el número de reuniones o *meetings*: siguen habiendo momentos en que los desarrolladores comparten su evolución personal en el desarrollo, pero son reuniones cortas. En algunos métodos ágiles incluso se limitan en el tiempo las reuniones.

- Si el mercado cambia, aparece un nuevo competidor o el producto necesita de un cambio de ruta, es mucho más fácil hacerlo, pues el producto es mucho más manejable. Como los ciclos son más pequeños, el cliente ve parte del producto mucho antes, y es más fácil evitar disgustos de última hora.

Algunas desventajas de los métodos ágiles:

- Es más difícil manejar proyectos grandes y se depende en gran parte del nivel de los desarrolladores. Es impensable seguir un método ágil con un equipo poco capaz.

- Requieres un equipo que sepa manejar el método, lo cual, aunque cada vez es más frecuente, no siempre está disponible.

4.4. Estrategia de desarrollo

En **Lectums** se ha seguido una estrategia ágil. Sin utilizar ningún método particular, el proceso ha sido el de añadir pequeñas funcionalidades cada vez y volver a empezar el proceso. Evidentemente no se ha podido llevar el método al máximo porque sólo había una persona –yo- desarrollando, pero no se ha seguido la línea de hacer grandes desarrollos sin comprobar cada poco tiempo los resultados.

¿Por qué he decidido seguir un método ágil? Fundamentalmente porque no me siento cómodo desarrollando sin tener feedback, necesito ver cómo las cosas van tomando forma. En el caso del desarrollo web, esto cobra, de hecho, mayor sentido, al ser un medio en el que es fácil ir viendo los resultados.

No creo en un proceso monolítico en que se tarde más en especificar y diseñar que en implementar. Creo que es en la etapa de desarrollo, y no en la de especificación, cuando realmente estás en contacto con la realidad y puedes entender el problema en mayor profundidad. Cualquier método que alargue la llegada de esta etapa no me parece especialmente productivo.

4.5. Testing

Al seguir un método ágil el testeo no ha sido una etapa final, sino una etapa constante en el desarrollo. Al final de cada pequeño ciclo, incluso por cada funcionalidad, se han revisado los posibles errores, corregido aquellos elementos que eran erróneos, e incluso modificado ligeramente el diseño para adaptarlo a la realidad. Ha sido un proceso constante de evaluación, lo que, bajo mi punto de vista, hace el software real desde el principio y de mayor calidad.

Los errores típicos que se pueden encontrar en un proyecto web se pueden dividir en 3 tipos fundamentales:

- Errores de programación: aparecen errores en tiempo de interpretación –PHP es un lenguaje interpretado- o simplemente las cosas no funcionan como deberían.

- Errores de maquetación: el paso del diseño en Photoshop a la web real tiene algún defecto, fuentes que no son correctas, colores, formas… o simplemente dos navegadores renderizan el diseño de formas diferentes lo cual, en algunos casos, no resulta aceptable.

- Errores de usabilidad: los usuarios que prueban el sistema encuentran problemas a la hora de utilizarlo o directamente no saben cómo hacerlo. Esto es algo muy frecuente y debe ser tratado.

En **Lectums** yo mismo he corregido todos los errores, mientras que amigos, familiares y mi tutor me han ayudado a detectar algunos, sobretodo en cuanto a los errores de usabilidad.

5. Viabilidad

5.1. Definiendo la necesidad

El libro probablemente constituya uno de los pocos medios tradicionales que no sólo conserva su volumen de ventas sino que en algunos casos las incrementa, especialmente si hablamos del **formato de libros digital o e-book**. Esto nos demuestra que el gusto por la lectura se conserva intacto en una buena parte de la población.

No obstante, el enorme volumen de libros que se puede encontrar hoy en día, en que acumulamos siglos y siglos de literatura, hace que las ventas se concentren en un reducido grupo de bestsellers que, probablemente, cuentan con un marketing agresivo que los diferencia del resto. **La dificultad para encontrar la aguja dentro del pajar, libros que realmente te puedan interesar dentro de la vorágine inmensa de lecturas posibles, es el problema que pretende solucionar Lectums.**

Además, existe también una costumbre muy importante que constituye la segundad necesidad –la primera es encontrar libros que nos interesen–, y es compartir nuestros gustos con los demás: esto es una afición humana aplicable a cualquier cosa, pero muy especialmente a la literatura. ¿Cuántas veces hemos hablado con amigos o familiares de los últimos libros que hemos leído? ¿Qué tal si en vez de con unos cuantos amigos o familiares, pudiéramos comentar nuestras experiencias literarias con cientos de usuarios de todo el mundo? A partir de este segundo concepto se plantea la idea de red social para amantes de la literatura.

La proliferación de redes sociales que estamos presenciando en este comienzo de siglo XXI nos indica que este tipo de servicios web no sólo son rentables, sino que constituye una de las mayores oportunidades de negocio que podemos encontrar en este momento en Internet. Algunos ejemplos, como **Facebook** o **Tuenti**, no han encontrado quizás el modelo de negocio correcto, entre otras cosas porque es difícil aplicar el modelo de la publicidad en este tipo de redes, y mucho más aún el de la compra de productos. No es el caso de **Lectums**, en que sí tiene mucho sentido la venta de libros, tanto en formato escrito como en formato digital.

Por todo lo dicho anteriormente, considero que existe un nicho muy claro que será el que aproveche **Lectums: un lugar en el que poder debatir e informarse sobre cualquier tipo de libro con cualquier usuario registrado en la red.**

5.2. Web 2.0

Para estudiar la web 2.0 y sus implicaciones lo primero que habría que tener en cuenta es, quizás, el propio origen del término “web 2.0”.

Según la Wikipedia, que por cierto es un gran ejemplo de web 2.0, esta palabra fue definida por primera vez por Tim O'Reilly, fundador y presidente de O'Reilly Media:

> *El término **Web 2.0** es asociado usualmente con Tim O'Reilly debido a la referencia hecha en la conferencia O'Reilly Media Web 2.0 en 2004. El término fue utilizado para referirse a una segunda generación en la historia del desarrollo de tecnología Web basada en comunidades de usuarios y una gama especial de servicios, como las redes sociales, los blogs, los wikis o las folcsonomías, que fomentan la colaboración y el intercambio ágil y eficaz de información entre los usuarios de una comunidad o red social.*

Dicha conferencia nació de una lluvia de ideas con Craig Cline de MediaLive.

Pero ¿qué es la web 2.0? La web 2.0 no es una tecnología, ni un lenguaje, ni siquiera un framework de desarrollo. La web 2.0 es una forma de entender Internet. Es una tendencia, un paradigma, un fenómeno que está revolucionando Internet y que es la evolución lógica de la web 1.0, forma de denominar a la famosa "*burbuja de las punto com*".

Durante la época de la explosión de los grandes portales, estos eran enormes amasijos de información, la mayoría en portada, que basaban su modelo de negocio básicamente en la publicidad. La caída en el rendimiento de esta última originó que inversiones multimillonarias quedaran en agua de borrajas y muchísimas empresas tuvieran que cerrar, algunas de bastante renombre.

Se entendía Internet como un medio sin interacción, prácticamente un libro cuya principal ventaja eran los hiperenlaces, pero en el que los visitantes no podían "participar". Estos últimos desarrollaron una habilidad especial que les permitía eludir fácilmente los banners publicitarios, sus cerebros se acostumbraron a ignorarlos. La publicidad agresiva perdió su razón de ser y cayó en el olvido.

A partir de la web 2.0 cambia la forma de entender el medio. Internet ya no es un gran libro inerte, sino que pasa a ser un medio en el que se puede interactuar; no sólo eso, sino que esa interacción pasa a ser la base de las páginas web, que se convierten en grandes aplicaciones similares a las de escritorio, que requieren mejores profesionales, buenos desarrolladores que sean capaces de acercar Internet a la funcionalidad de un escritorio de PC convencional.

Y no sólo buenos programadores, sino grandes diseñadores capaces de estudiar detalladamente cómo tiene que ser la interfaz de usuario, la estructura de la información, el diseño gráfico, para que los usuarios aprendan fácilmente cómo interactuar con estas nuevas aplicaciones web.

La arquitectura de la información, es decir, cómo se han de ordenar los datos para que el usuario encuentre fácilmente lo que busca, pasa a cobrar mayor importancia con la web 2.0, ya que entendemos que el usuario es el objetivo de nuestra página web, y ha de ser este último el que encuentre utilidad a nuestra aplicación.

De aquí surge la figura del arquitecto de la información, que estudia la mejor forma de organizar todo el contenido de la aplicación, siguiendo varias reglas tanto lógicas como empíricas.

Destacar, por ejemplo, la regla de los 5 puntos, que implica que una lista de secciones no debería superar nunca los 5 elementos, y la regla del 3, que implica que un menú no debería jamás más de 3 niveles de jerarquía.

Como la web ha pasado de ser un medio estático a necesitar de una interacción con un agente externo –el usuario– hemos de garantizar que este último entiende perfectamente cómo funciona nuestra aplicación. De aquí el concepto de usabilidad, es decir, la capacidad de nuestra aplicación para ser utilizada con facilidad.

La curva de aprendizaje, como en cualquier otro campo, define la cantidad de tiempo que necesitará un usuario para aprender a utilizar nuestra aplicación. En el caso de Internet, esto es especialmente crítico, ya que el usuario medio emplea menos tiempo en aprender a utilizar una aplicación de Internet que el que emplea en cualquier otra cosa del mundo real, pues tiene muchas más alternativas que quizás sean más sencillas.

La accesibilidad de un sitio web define si personas con algún tipo de discapacidad o con equipos poco convencionales van a poder acceder a nuestra aplicación y utilizarla de la misma manera que el resto de visitantes. Es especialmente importante si pretendemos que nuestra aplicación sea utilizada por el mayor número de personas posible.

Semántica es significado. En la web 2.0 se intenta que el contenido sea algo más que montones de palabras una tras otra. Se pretende que los programas de ordenador sean capaces de entender esa información, de darle contexto.

Para ello, se añade a la información, en el código de la página, etiquetas XHTML que agregan ese contexto que falta, esas relaciones entre conceptos que los ordenadores desconocen a priori.

Desde luego, se trata de uno de los campos más complejos, pues forma parte de la Inteligencia Artificial, una de las áreas más complejos del software. Quién sabe si quizás, algún día, un agente artificial pueda entender la información contenida en Internet y ayudarnos con nuestros deberes diarios.

El marketing estudia la forma de "convencer" al visitante de que debe adquirir nuestro producto. En esto el concepto de marketing no varía del habitual. Sin embargo, la forma de conseguir esto varía significativamente en Internet respecto al mundo real. El marketing relacional toma en Internet una importancia crucial a partir de la aparición de la denominada web 2.0.

Bajo estas premisas, empiezan a surgir multitud de proyectos 2.0 que pretenden llenar vacíos en el mercado, ocupar ciertos nichos que pueden resultar rentables en un momento dado. En este contexto pretende aparecer Lectums: cumpliendo los puntos que definen la web 2.0 y, por tanto, pudiendo defender que se trata de un proyecto web 2.0 que nada tiene que ver con los antiguos portales. Sufrirá los problemas de las nuevas formas de desarrollar de la misma forma que se aprovechará de las nuevas ventajas que de ello se deriva.

5.3. SaaS como modelo de servicio

Con la aparición de la web 2.0 y de la nube –aplicaciones que guardan los datos de sus clientes en servidores alrededor del mundo– el concepto de SaaS se ha extendido y evolucionado de forma vertiginosa. SaaS son las siglas de *Software as a Service*, y consisten en ofrecer servicios de software sin necesidad de que el cliente instale la aplicación, pues es una aplicación web a la cual puede acceder con un navegador y, además, los datos se guardan en un servidor.

Ahora las tarifas ya no están tan basadas en el producto en sí, como en el uso que el cliente haga de él, de forma que se paga más por el uso del servidor –una mensualidad– que por el software en sí, que es gratuito en la mayoría de los casos.

Lectums es, por supuesto, SaaS, aunque no basará su modelo de negocio en tarifas por uso, sino en otras fuentes de ingresos que no recaerán en el usuario final.

5.4. Idea de negocio

La idea de negocio, al menos en su etapa inicial, se basa en la publicidad y la venta de libros por sistema de afiliación. Es decir, se colocará en la red publicidad, en un futuro inteligente, por la que las empresas que se publiciten pagarán una cierta cantidad de dinero. Alternativamente, y no menos importante, las páginas con información sobre libros tendrán enlaces a empresas que vendan esos libros, ganando Lectums, por cada venta, un cierto margen.

5.4.1. Ingresos

Es muy difícil calcular los ingresos teóricos de Lectums. No obstante, si suponemos 3 banners publicitarios a 500 euros mensuales, unas ventas del 1% (1 venta por cada 100 visitas), y un margen medio de 2 euros por libro vendido, podemos estimar los siguientes ingresos para 300.000 visitas mensuales:

Concepto	**Cantidad**	**Precio (€)**	**Total (€)**
Banners	3	500	1500
Libros vendidos	3000	2	6000

Tabla 3 Ingresos estimados

El total es de 7.500€ mensuales, esto es, 90.000€ anuales. Por supuesto es una suposición bastante alegre, habría que hacer un estudio de mercado mucho más estricto.

5.4.2. Gastos

Los gastos se basan sobretodo en gastos de servidor y de autónomo, además de un salario base mínimo deseable. Respecto a los primeros, contaremos 100€ mensuales con ese flujo de visitas. Los autónomos, supongamos 500€ para una cotización decente; un salario mínimo deseable podría establecerse en 1000€, más un 40% en concepto de gastos a cargo de la empresa, esto es, 1400€.

Concepto	Cantidad	Precio (€)	Total (€)
Autónomos	12	500	6000
Servidor	12	100	1200
Salario	14	1400	19600

Tabla 4 Gastos estimados

El total es 26.800€ anuales. Para un inicio, valdría un servidor mucho más humilde. Se puede ver una comparativa de este tipo de servidores en el Apéndice D.

5.4.3. Totales

Suponiendo un 35% de Impuesto de Sociedades sobre 63.200€ (90.000 – 26.800), el total neto es 41.080€ anuales.

5.5. Licencia

En este proyecto no tiene importancia la licencia, puesto que el software no se va a distribuir, pues no es este el modelo de negocio. En el caso de materiales ajenos a la Lectums, se aplican las licencias que les correspondan.

5.6. Primeros pasos

El primer paso que debería seguir si quiero que el proyecto Lectums no sea un proyecto más en un cajón es crear más contenido para, en un momento dado, estar preparado para lanzarlo y aprovechar el tirón del *happy month* –el primer mes en Google registra un sitio web en su directorio potencia los resultados de búsqueda para ese dominio, como una forma de *ayudar a los nuevos*.

Hecho esto, habría que comprobar si la idea funciona o no, pero no sería hasta varios meses después cuando se podría empezar a ver resultados. Si funcionara, sería el momento de contactar con los distribuidores de libros y las editoriales para poder llegar a acuerdos que permitieran poner en práctica el modelo de negocio.

6. Marketing

6.1. Introducción

Las diferentes estrategias de marketing que se utilicen en Lectums deben estar muy meditadas de forma que se obtenga el mejor resultado, especialmente al principio. Al ser este un proyecto muy largo y complejo, queda fuera de este Proyecto de Fin de Carrera implementar la mayoría de estrategias, pero sí que creo interesante definirlas de cara a poder implementarlas en el futuro.

6.2. Estrategias SEO

6.2.1. ¿Qué es SEO?

SEO es el acrónimo de Search Engine Optimizer, o "Optimización de los Motores de Búsqueda". Se puede referir tanto a la persona encargada de ello, como a la "ciencia" en sí. Según la Wikipedia:

> *El posicionamiento en buscadores o posicionamiento web es el resultado de la recuperación de información en la base de datos de los grandes motores de búsqueda de Internet por el uso de algoritmos de búsqueda en el software.*
>
> *La tarea de ajustar la información de las páginas que se pretenden hacer aparecer en primeras posiciones de los resultados es conocida como SEO, sigla en inglés de* Search Engine Optimization*, o sea, 'Optimización para motores de búsqueda'. Consiste en aplicar diversas técnicas tendientes a lograr que los buscadores de Internet sitúen determinada página web en una posición y categoría alta (primeras posiciones) dentro de su página de resultados para determinados términos y frases clave de búsqueda. También es entendido como las técnicas de desarrollo web que tengan como objetivo mejorar la posición de un determinado sitio web por sus páginas en la lista de resultados de los motores de búsqueda que en inglés se identifican como SERPs y forma parte de la jerga de los webmasters hispanohablantes.*

Dicho de otra manera, a la tarea de modificar una página web para obtener mejores posiciones en los motores de búsqueda se le llama SEO.

El SEO puede suponer la diferencia entre el éxito y el fracaso de cualquier proyecto web, ya que el tráfico en Internet viene, en su mayoría, de los buscadores y, por supuesto, Google es el líder en este campo. Por lo tanto optimizar **Lectums** para que obtenga mejores resultados en los motores de búsqueda y en especial en Google, es clave, y voy a dedicar varias páginas de la memoria en explicar las diferentes técnicas que utilizaré.

6.2.2. Técnicas básicas

6.2.2.1. El contenido manda

Lo primero que hemos de tener claro a la hora de elaborar una buena estrategia SEO, es que el contenido de nuestra página web es lo más importante a la hora de posicionar en los buscadores. El motivo es muy sencillo, los buscadores rastrean nuestra página web en busca de palabras clave; cuanto más contenido tengamos, más palabras clave encontrará el motor de búsqueda. Este es, quizás, el motivo fundamental de que la Wikipedia esté la primera en los resultados de búsqueda de casi cualquier cosa.

Ahora bien, ¿se ha de poner cualquier cosa con tal de aumentar el contenido de la página web? La respuesta es un no rotundo, por varios motivos. Primero, porque los buscadores cuentan con agentes humanos, que visitarán la web y nos penalizarán si ven que el contenido que tenemos está pensando para los buscadores y no para las personas. Segundo, porque no vale con posicionar bien en los motores de búsqueda: queremos que los visitantes que lleguen a nuestra página web encuentren lo que buscan; en caso contrario, sentirán frustración, hablarán mal de nuestra página web, y no conseguiremos nuestros objetivos.

Conclusión: crear mucho contenido, relacionado con tu página web, y de mucha calidad, es un paso fundamental para lograr que tu estrategia SEO sea realmente efectiva.

6.2.2.2. Enlaces entrantes y enlaces salientes

Los buscadores tienen muy claro lo que es la web: la web es un montón de documentos conectados entre ellos por hipervínculos. Y, por ello, valoran enormemente que enlaces a otras páginas y que otras páginas te enlacen a ti.

En concreto, los enlaces entrantes marcan en Google lo que se denomina como PageRank. El PageRank es uno de los factores que hacen que tu página web esté más arriba o más abajo en los resultados de Google; y el PageRank se forma a partir de cuántos enlaces entrantes tienes, es decir, cuánta gente enlaza a tu web, y qué calidad tienen esos enlaces, es decir, cuánto PageRank tienen las páginas web que te enlazan.

En general, es mejor que te enlace una página con mucho PageRank que diez con muy poco. De hecho, los enlaces basura, enlaces de muy poco valor, podrían llegar a penalizarte. Conseguir una cantidad respetable de enlaces que provengan de páginas web con alto PageRank es otra de las claves de una buena estrategia SEO.

6.2.2.3. Maquetación

Una correcta maquetación de la página web es muy importante a la hora de posicionar en los buscadores. ¡Hay que utilizar las etiquetas para lo que fueron creadas! Por ejemplo, un único H1 por página, en el que esté incluida la idea fundamental de esa página.

Siempre hay que utilizar una etiqueta *title*, que podría estar formada por "título de la página | nombre de la web".

Hay que procurar utilizar sólo las etiquetas XHTML imprescindibles. Cuanto más ligero y sencillo sea el código de la maqueta, más sencillo va a ser para los buscadores aislar el contenido y mejor se van a posicionar las páginas.

6.2.2.4. Peso de la página

Las últimas informaciones indican que Google valorará el peso de las páginas y la rapidez de carga de estas a la hora de posicionarlas en su buscador. Incluso aunque no fuera así, que tu página web cargue rápido es fundamental para que los visitantes no se vayan de ella. Hay que tenerlo siempre en cuenta.

6.2.2.5. Palabras clave

Hay que procurar siempre tener claras qué palabras definen claramente el contenido de la página que estás haciendo. Es importante que en el texto de esa página se repitan algunas veces, aunque sin abusar. La idea es que los motores de búsqueda tengan claro de qué va la página web.

A la hora de definir estas palabras clave, hay que pensar en cómo buscarías tú la página que estás haciendo en un buscador. Hay que procurar que las palabras que usarías sean las que estás posicionando.

6.2.3. La importancia del contenido

He hablado de varias técnicas a la hora de posicionar un sitio web en los buscadores. Decía que tenemos que trabajar el contenido de forma que sea fácil de encontrar por los buscadores, sin perder el toque humano, que es lo que diferencia a un texto de un conjunto inconexo de palabras clave. De hecho, he insistido en que los buscadores tienen editores humanos que encuentra fácilmente la diferencia. Pero, ¿qué de la importancia del propio contenido?

6.2.3.1. Cantidad

No nos engañemos. Cuanto más contenido tengamos, más tráfico vamos a atraer de los buscadores. Cada porción de contenido es una puerta por la que puede, o no, entrar alguien; cuantas más puertas, más posibles entradas tendremos. Y esto es una máxima que demuestra cada día la Wikipedia.

Por lo tanto, y aquí hay un gran trabajo, para aumentar el tráfico hacia nuestras páginas web hemos de construir mucho contenido, y con mucha frecuencia.

6.2.3.2. Calidad

¿Da igual si el contenido es bueno o no? De ninguna manera; recordemos nuevamente que nuestro contenido no se orienta a buscadores, sino a personas. Si escribimos mal, con faltas de ortografía, sin signos de puntuación, sobre temas que no conocemos o diciendo cosas que no son correctas, los visitantes llegarán, pero no volverán. Tendremos visitas vacías sin ningún valor.

6.2.3.3. Originalidad

Si hablas de un tema que se repite infinitamente en Internet, no vas a atraer demasiadas visitas. Por muy bien que esté explicado o por muy bien que intentes posicionar el texto. Sencillamente la competencia es demasiado alta. Hay que intentar hablar de temas de los que nadie antes haya hablado o, al menos, con un enfoque diferente. La única posible excepción es el principio, cuando lanzas tu proyecto: en este caso quizás sea interesante hablar de temas trillados, porque de esta forma se consigue una base de contenido y la página tiene cierta coherencia; que un tema se haya repetido hasta aburrir no implica que no sea básico.

6.2.3.4. Oportunismo

El contenido no tendrá siempre el mismo volumen de tráfico. Hay momentos en que el tráfico aumenta, como se puede ver fácilmente en Google Trends. Esto es debido a que un contenido determinado no tiene el mismo interés a lo largo del tiempo. Hablar, por ejemplo, de Michael Jackson en el momento en que falleció genera mucho más tráfico sencillamente porque a la gente le interesa más.

El problema es que todos los desarrolladores saben eso, y hablar de las noticias del día no genera mucho tráfico porque todo el mundo habla de ello. ¿Qué hacer entonces? Anticiparse: si sabes que un determinado evento va a ocurrir, genera contenido sobre él, para así irte posicionando mientras no hay mucha competencia. Cuando ese evento suceda, el tráfico será muy alto porque tú ya estarás arriba.

6.2.3.5. El contenido en Lectums

En **Lectums** el contenido fundamental es la base de datos de libros y autores. Esto representa muchas páginas diferentes que los buscadores devorarán con avidez. Hay que asegurarse, además, de que el contenido sea útil para los usuarios.

Publicar información sobre libros recién lanzados aporta la dosis de oportunismo que comentaba antes, y puede hacer que durante un tiempo entren nuevos visitantes.

6.2.4. Construir buenos enlaces

Como ya sabemos los enlaces, tanto internos como externos, son básicos a la hora de conseguir posicionar bien un sitio web, pues van a determinar, entre otras cosas, el PageRank; además, otros webmasters conocerán de la existencia de tu sitio web al ser enlazados, y podrían devolvernos al favor. Y eso sin contar con que, precisamente, el sentido de la web es que es una red, es decir, un conjunto de nodos enlazados: estamos contribuyendo al desarrollo de ésta.

Ahora bien, hay formas y formas de hacer enlaces. "Haz clic aquí" no es un buen enlace. ¿por qué?

Cada vez más la web semántica toma relevancia y desplaza a la antigua web 1.0. Si antiguamente los motores de búsqueda sólo se fijaban en si tus términos de búsqueda coincidían con las palabras de los sitios web indexados, ahora cada vez más intentan "entender" el contenido de sus fuentes para poder servir resultados más eficientes. Crear enlaces semánticos ayuda a esa tarea y posicionas mejor. Hay que seguir, pues, los puntos indicados a continuación:

- Texto del enlace: nada de "Haz clic aquí", o "Pincha aquí", o "leer más después del salto" -este último muy típico en blogs. El texto del enlace ha de describir la página de destino. Si queremos enlazar a la página web de un restaurante, un enlace del tipo "Restaurante fulanito, las mejores comidas" es mucho mejor que los anteriores ejemplos.

- Enlaces en negrita: los enlaces deberían ser palabras clave; antes dije que las palabras clave en negrita están mejor consideradas. Conclusión: enlaces en negrita.

- Atributo *title*: ¡muy importante! El atributo *title* añade aún más semántica. Si se hizo bien el texto del enlace, el atributo *title* podría ser el mismo que el texto.

- No hay que enlazar a páginas dudosas ni mucho menos a granjas de enlaces; aunque puede resultar tentador, no van a dar buen resultado e incluso nos podrían *banear*.

6.2.4.1. Los enlaces en Lectums

En **Lectums** hay pocos enlaces hacia fuera, excepto quizás en alguna FAQ. No obstante, los enlaces interiores tienen la misma importancia. En este caso, habrá que tener cuidado con los textos y la forma en que enlazo.

6.2.5. URL's amigables

Los motores de búsqueda como Google se basan básicamente en dos cosas: palabras clave y semántica. Y, de hecho, cada vez menos en la primera y más en la segunda. Pues bien, demos un paso más, y estudiemos ahora la propia URL del sitio.

Los motores de búsqueda se fijan en las URL's buscando las palabras clave que el usuario ha introducido; eso incluye el dominio. Es decir, que si hemos decidido que la palabra clave más importante de la nueva página web es, digamos, *gatos*, sería muy interesante que ésta apareciera en el dominio. Y puedo demostrarlo: si buscamos gatos en Google, aparece la primera -a día de hoy- mundogatos.com.

Pues bien, los motores de búsqueda no se quedan en el dominio. Siguen a partir de ahí. Lejos quedó el tiempo en que "dominio.com/index.php?id=1″ era una url aceptable. Ahora, además, hay que añadir nuestras palabras clave en la URL, sustituyendo ese index.php?id=1 por cosas con sentido, con semántica. Por ejemplo, si estamos hablando de comida para gatos, y siguiendo con el ejemplo de mundogatos.com, deberíamos construir una URL del tipo mundogatos.com/comida-para-gatos.

¿Cómo hacemos esto? Una opción poco recomendable es la obvia, la que primero se nos viene a la cabeza: creamos un archivo html con el nombre "comida-para-gatos.html", o una carpeta con el nombre "comida-para-gatos". Es una opción, pero *mod_rewrite* nos permite hacer cosas más inteligentes, y los frameworks MVC todavía más.

En cualquier caso, la clave está en cuidar las URL's.

6.2.5.1. URL's en Lectums

En Lectums las URL's están muy pensadas, haciendo un uso intensivo de las herramientas que me proporciona Codeigniter. En cada una de ellas hay información semántica sobre el contenido que hay en la página y el formato habitual es:

http://www.lectums.com/modelo/read/titulo-del-contenido/id-del-contenido

6.2.6. El tamaño de la página web

Hace un tiempo empezó a correr el rumor de que Google iba a tener en cuenta el peso de la página a la hora de indexarla en sus servidores, y empezó a cundir el pánico en lo que, en el fondo, debería ser una cuestión solucionada a priori. Me explico: a veces le damos tanta prioridad a los buscadores que olvidamos que estamos desarrollando páginas web para personas y, para el caso que nos ocupa, a las personas les gusta que la página cargue rápido.

Para que la página cargue rápido se deberían cumplir una serie de puntos, entre los que destacan:

- Procurar que las imágenes ocupen lo menos posible, utilizando formatos como PNG. Guardar las imágenes siempre en el tamaño adecuado, no más grandes para luego reducirlas en tiempo real, porque esto ralentiza enormemente la carga. Utilizar un CDN puede ser una buena idea si contamos con los recursos necesarios.

- Utilizar un marcado XHTML correcto. Nada de tablas, generan un código enorme e innecesario. Una maquetación bien pensada con un código CSS ligero que aproveche su condición de cascada ayudará mucho.

- Optimizar al máximo los scripts, tanto los del lado del servidor como los del lado del cliente.

- Utilizar un sistema de caché siempre que sea posible. No obstante, esto requiere un control de cambios, de forma que la página no quede inmutable para siempre.

6.2.6.1. El tamaño de Lectums

En **Lectums** se seguirá una maquetación pensada desde el principio para ser ligera a la par que usable, de forma que se reduzcan los tiempos de carga. Un sistema de cache proporcionado por Codeigniter debería permitir reducir los tiempos de carga. Por supuesto, la base de datos debe de estar correctamente optimizada, como se explica en el apartado correspondiente.

6.3. Publicidad

Hay diferentes tipos de publicidad que se pueden aplicar a **Lectums**, y se pueden dividir en dos grupos obvios: publicidad gratuita y publicidad pagada:

6.3.1. Publicidad gratuita

- SEO: el posicionamiento en los buscadores es gratuito y es la mejor estrategia de marketing gratuita que existe.

- Acuerdos de reciprocidad con otras páginas del sector, según los cuales se muestre cierta publicidad en **Lectums** a cambio de la promoción del proyecto en esas otras páginas.

- Participación en foros y blogs del sector del libro, para dar a conocer el proyecto entre los que puedan estar interesados.

- Campañas offline, aunque estas últimas quizás no tengan mucho sentido dado el poco impacto que tendrían.

6.3.2. Publicidad de pago

- Pago en portales del sector del libro a cambio de publicidad. Se deberían escoger audiencias muy segmentadas, de forma que la tasa de retorno –la relación entre la inversión y el número de conversiones- sea mayor. Una conversión en **Lectums** podría ser un usuario registrado o la compra de un libro.

- Publicidad contextual de Google. Se puede invertir un determinado presupuesto en palabras clave de Google, de forma que la publicidad de **Lectums** aparezca para determinadas búsquedas.

- Acuerdos con editoriales que permitan la publicación de más contenido, lo que puede llevar aparejado un aumento del tráfico y de las conversiones.

6.4. Redes sociales

Hay diferentes estrategias a seguir en cuanto a las redes sociales generalistas a la hora de conseguir maximizar la audiencia de **Lectums**. Creo que hay dos redes sociales a destacar: Twitter y Facebook.

- **Twitter:** tiene carácter profesional, por lo tanto habría que dejarlo para libros de este tipo o contactos esporádicos con algún profesional del sector. No obstante, puede ser un gran método de incrementar mi red de contactos de cara a obtener acuerdos interesantes.

- **Facebook:** en el futuro es casi imprescindible integrar el sistema de login de Facebook con **Lectums**, de forma que cualquier usuario de Facebook pueda entrar inmediatamente y sin esperas a **Lectums**. De esta forma, además, los usuarios podrían compartir en el muro los libros que les interesaran más, de forma que se incrementaría el tráfico de **Lectums**. Creo importante, además, crear un grupo para **Lectums**, de forma que se cree comunidad y credibilidad para la plataforma.

7. Balances y conclusiones

7.1. Planificación final y análisis económico

El recuento final de horas es ligeramente superior a lo previsto, pero bastante cercano. La diferencia, seguramente, estriba en la repartición de horas entre los distintos roles. La tarea de implementación ha sido más ligera de lo que estimaba, mientras que la de maquetación ha sido un poco superior. El testeo ha resultado bastante sencillo, mientras que el trabajo de introducir contenido ha sido especialmente tedioso.

De media, he trabajado durante 5 meses unas seis horas diarias de lunes a viernes, lo que suma un total de 600 horas (estaban estimadas 535). Esta diferencia, como indicaba, se debe sobretodo al trabajo de maquetación, según se ve en la siguiente tabla:

Profesional	**Horas estimadas**	**Horas finales**	**Total**
Analista	75	90	90
Programador	150	160	250
Maquetador	100	140	390
Diseñador web	60	60	450
Documentador	50	50	500
Introductor de contenido	100	100	600

Tabla 5 Planificación final

Debido a esto, el análisis económico no varía especialmente, ya que las horas totales son parecidas y los sueldos del programador y el maquetador son parecidos también. En general, creo que se han alcanzado los objetivos propuestos. Valga decir que son datos orientativos y que pueden haber pequeñas variaciones.

7.2. Conclusiones y propuestas de mejora

Lectums ha sido una gran experiencia para mí tanto por el desarrollo en sí como por el hecho de medir fuerzas y enfrentarme a un proyecto de cierta envergadura. Pese a haber participado en muchos proyectos web en mi vida profesional, nunca lo había hecho con un proyecto tan grande y en el que yo tuviera que manejar todo el conjunto.

Ha sido, además, la constatación, por mi parte, de la potencia del paradigma MVC y, probablemente, el proyecto en el que más he consolidado mis conocimientos al respecto. Por supuesto, también he aplicado nuevas técnicas de maquetación, usabilidad y accesibilidad que me han permitido profundizar en el campo del diseño web; pese a no ser el diseño web un campo propio de la ingeniería, considero interesante el haber adquirido conocimientos al respecto que me

permiten, en cierta manera, encarar proyectos y comprender mucho mejor las necesidades del resto de profesionales que componen un equipo de desarrollo. Creo que es importante para un profesional, el Ingeniero Informático, que se supone debe estar al frente de proyectos de desarrollo, la comprensión del trabajo y las necesidades de todo el equipo.

He aprendido, además, técnicas de seguridad web que, aunque **Lectums** no requiera de un nivel demasiado estricto pues no contiene datos de alta sensibilidad (nivel 1 según la LOPD), sí requiere cierta atención al respecto.

Ha sido interesante, además, diseñar un pequeño plan de marketing y una hoja de ruta que me permitirán, si decido -como es mi intención- publicar el proyecto, entender mejor el medio y ser capaz, quizás, de rentabilizarlo.

Para mí ha sido muy importante haberme enfrentado a un proyecto de estas características con la presión del tiempo como garante del progreso continuado y haber conseguido cumplir los objetivos marcados que, aunque obviamente no dan por finalizado el proyecto **Lectums**, sí son un gran punto de partida. Y no lo dan por finalizado porque, evidentemente, **Lectums** está más allá de las posibilidades temporales de un proyecto de fin de carrera pero, no obstante y como digo, sí se han cumplido unos hitos más que razonables.

Echando la vista atrás y pensando en los objetivos iniciales debo decir que resulta mucho más costoso de lo que había pensado inicialmente el alcanzar las metas que me había planteado. Sí he creado una red social para amantes de la lectura, sí he esbozado un plan de marketing dedicado a su posible promoción en el momento de publicarlo, pero estoy lejos aún de implementar una estrategia que me permita rentabilizar el proyecto. Soy consciente, además, de que en los últimos meses han salido al mercado más de una docena de competidores que antes no sólo no existían sino que ahora forman parte de un nicho anteriormente vacío. Esto puede suponer una dificultad quizás insalvable para la rentabilización del proyecto. No obstante, esto me demuestra que mi idea no era descabellada o que, como mínimo, no soy el único al que se le ha ocurrido.

¿Qué podría añadir? Mil cosas. Una red social como esta tiene un potencial prácticamente ilimitado; se me ocurren cosas como integrar Facebook en **Lectums**, al menos la parte de conexión de usuarios, muros, etc. Se podría integrar algún sistema de base de datos con datos de bibliotecas públicas españolas con el objetivo de poder saber si cerca de ti –mediante geolocalización- hay alguna biblioteca que tenga el libro que te gusta; un sistema de *lista de deseos*, que permita a tus amigos comprar los libros que quieres. El sistema de compra a través de afiliados de forma que un usuario pueda comprar fácilmente un libro. Un sistema de amigos, con el que los usuarios puedan definir qué otros usuarios son amigos suyos –importable desde Facebook. Integración con Google Books… Realmente las posibilidades son infinitas.

Falta mucho para eso; no obstante, estoy satisfecho de haber llegado hasta aquí. Creo sinceramente que el proyecto **Lectums** es un proyecto del que sentirme orgulloso, no sólo por lo aprendido, sino también por el resultado final.

8. Apéndices

8.1. Apéndice A: Manuales

8.1.1. Manual de usuario

8.1.1.1. Acceso al sistema

Lectums se encuentra en un estado de beta privada. Por ello, deberás conseguir del administrador una cuenta de usuario para poder acceder. Éste te proporcionará un usuario y una contraseña. Una vez los tengas, debes acceder a la página principal:

http://www.lectums.com

Aparecerá la siguiente pantalla en la que tienes que introducir tu usuario y tu contraseña:

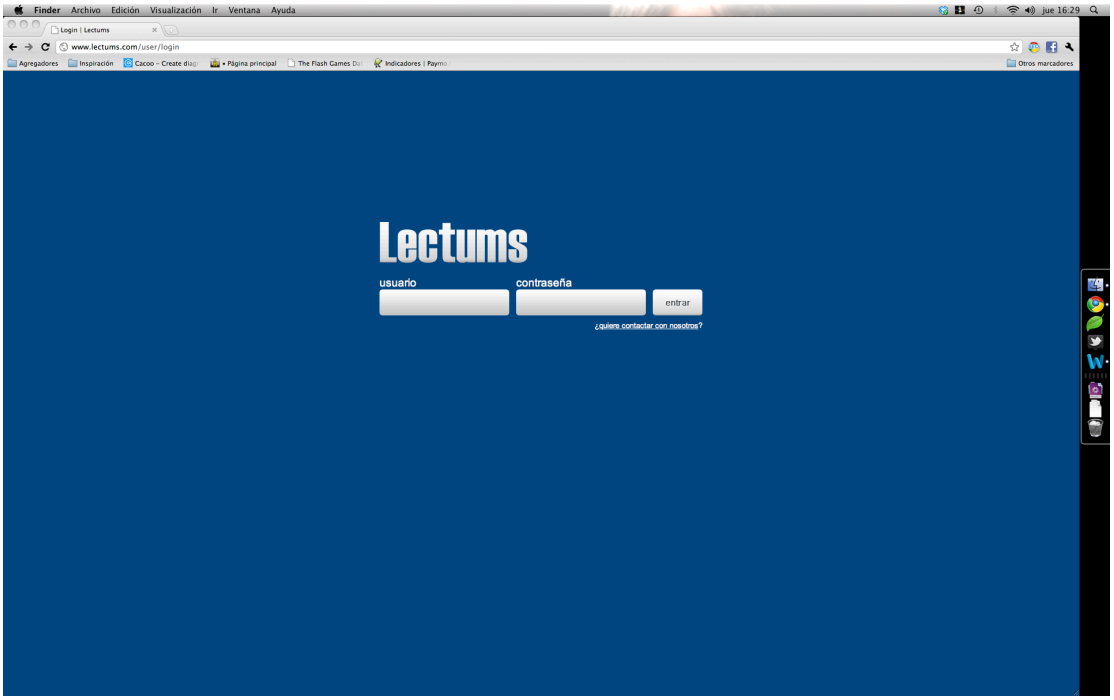

Ilustración 27 Pantalla de login de Lectums

Una vez introducidos, pulsa la tecla *enter* o haz clic en entrar. Si desearas ponerte en contacto con el administrador, el enlace *"¿quieres contactar con nosotros?"* es una buena opción.

Una vez dentro, debería aparecer una pantalla similar a esta:

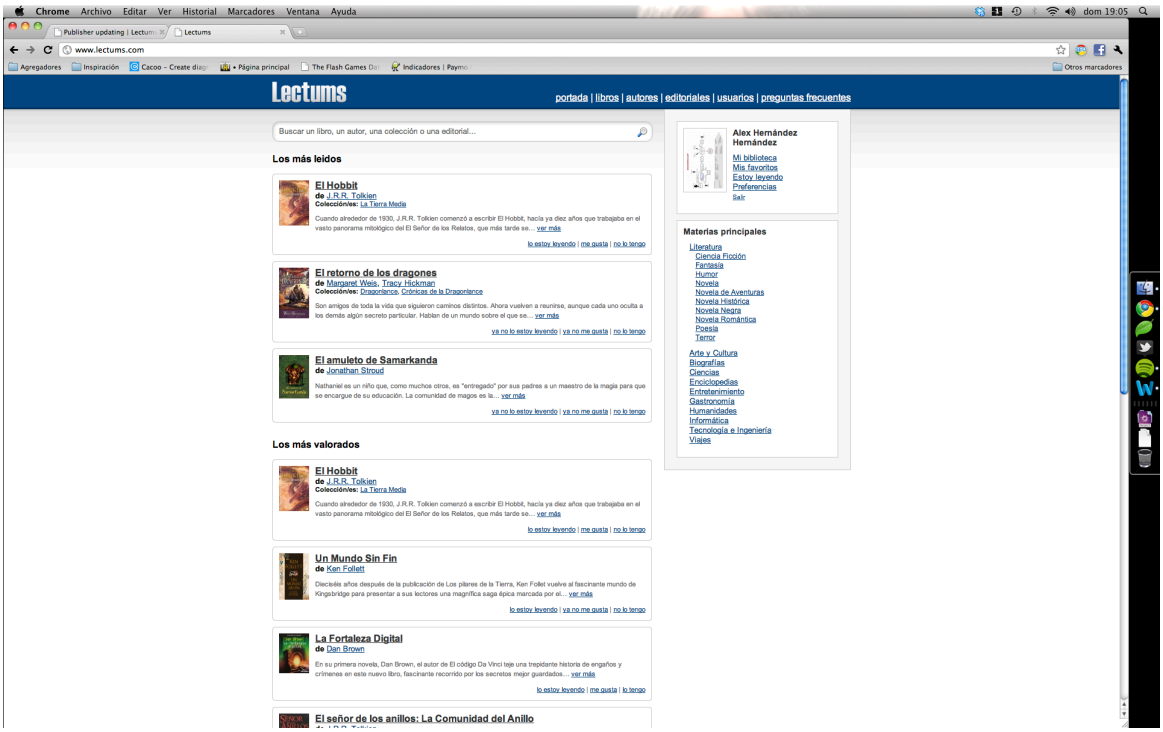

Ilustración 28 Página principal de Lectums

Con la diferencia de que en la parte de la derecha aparecerá información sobre tu propio usuario.

8.1.1.2. Preferencias

Modificar tus preferencias es lo primero que deberías hacer para disfrutar de una experiencia plena en **Lectums**. Para ello, pulsa en el enlace titulado *Preferencias*, en la parte de usuario de la columna derecha, en la zona superior. Aparecerá la siguiente pantalla con tu información:

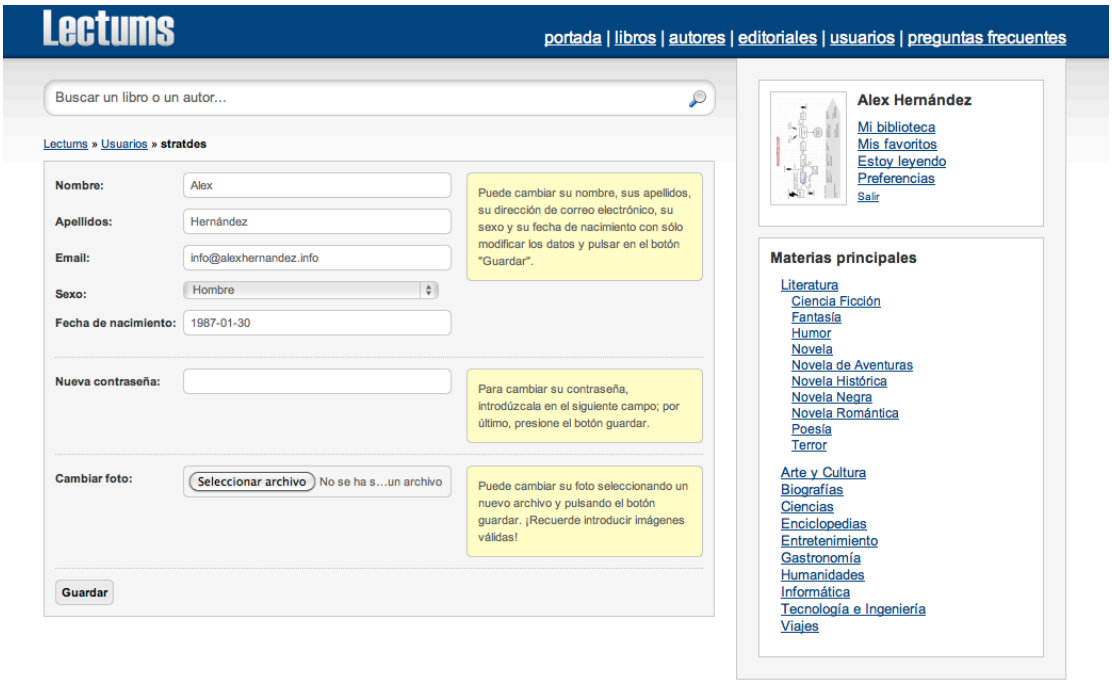

Ilustración 29 Preferencias de perfil

En el primer bloque de opciones tienes tus datos personales básicos, nombre, apellidos, dirección de correo electrónico, sexo y fecha de nacimiento. Modifica estos datos según tu situación personal.

En el siguiente bloque puedes introducir una nueva contraseña; es muy importante que efectúes este paso, ya que el administrador te proporcionará una contraseña aleatoria difícil de recordar.

Por último, en el tercer bloque, puedes seleccionar un archivo de imagen de tu disco duro que te servirá como foto; esta imagen aparecerá en la columna derecha, zona superior, en la parte de usuario, y cada vez que hagas un comentario de un libro o introduzcas un comentario en el muro de otro usuario.

Para guardar los cambios, pulsar Guardar.

8.1.1.3. Libros

Puedes consultar un listado completo de los libros de **Lectums** pulsando en el enlace *"Libros"*, en el menú superior. Verás un listado similar a este:

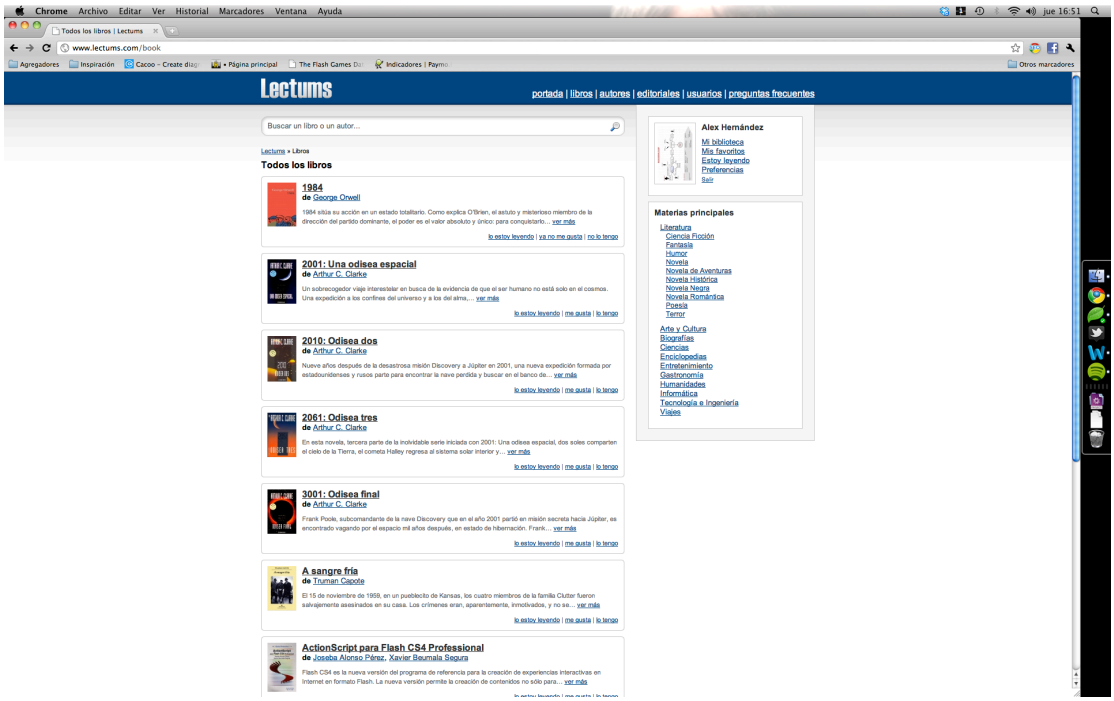

Ilustración 30 Listado de libros

Presta especial atención a las tres opciones que encontrarás en la parte inferior de cada libro, que son:

- **Lo estoy leyendo:** si clicas en este enlace estarás diciendo al sistema que estás leyendo ese libro. A partir de ese momento, podrás verlo en la sección *"Estoy leyendo"*. Si te has equivocado, o si acabas de leerlo, puedes pulsar en el botón *"Ya no lo estoy leyendo"*, que habrá aparecido en lugar del primero.

- **Me gusta:** si clicas en este enlace añadirás este libro a tu sección *"Favoritos"*. Si te has equivocado, o si descubres que ese libro en realidad no te gustaba, puedes pulsar el botón *"Ya no me gusta"*, que habrá aparecido en lugar del primero.

- **Lo tengo:** si clicas en este enlace, verás este libro en tu *"Biblioteca"*. Si te has equivocado o te has desprendido de este libro, puedes clicar en el enlace *"Ya no lo tengo"*, que habrá aparecido en lugar del primero.

Puedes, además, clicar en el autor, en la colección o en la editorial para ir a sus respectivas páginas. Si clicas en la página del libro, podrás ver su ficha completa, tal como se puede ver en esta página de ejemplo del libro *1984*:

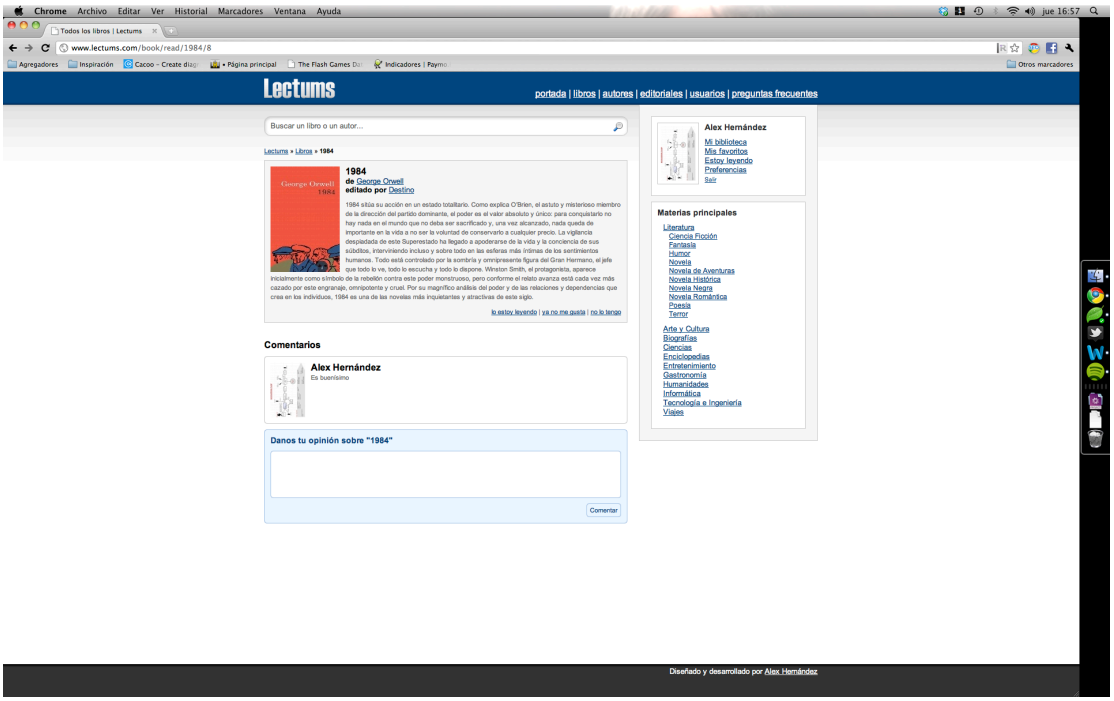

Ilustración 31 Ficha de libro

Puedes comentar este libro, dándonos tu opinión sobre el libro si escribes en el campo de la zona inferior y pulsas en el botón *"comentar"*. Por favor, ten en cuenta que este es un portal literario, deberías ser lo más cuidadoso posible y mantener las normas de respeto más elementales; sino, tu comentario puede ser borrado.

8.1.1.4. Categorías

Puedes ver el árbol de categorías en el área derecha de la página. Las categorías te ayudan a buscar entre los muchos libros disponibles en **Lectums**, según la materia de la que tratan. Si haces clic en cualquiera de ellas, podrás ver una pantalla como la siguiente:

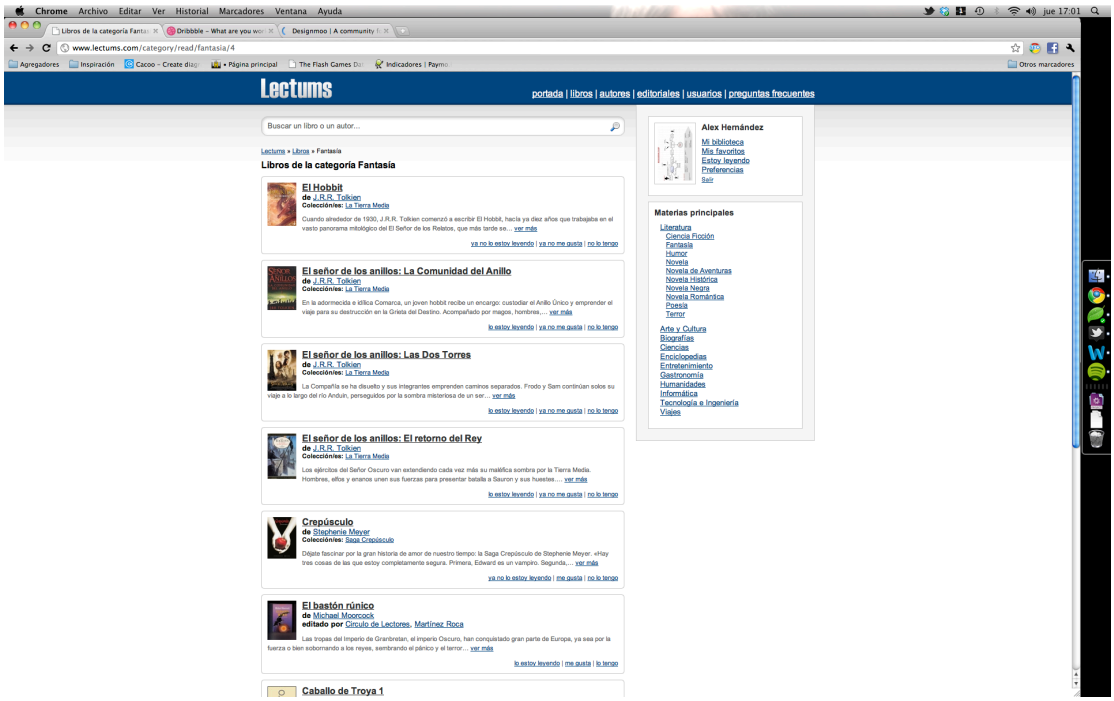

Ilustración 32 Categorías

Este listado es igual que el listado disponible en la sección *"Libros"*, con la diferencia de que sólo salen libros, en este caso, de la categoría *"Fantasía"*.

8.1.1.5. Autores

En la sección de autores puedes encontrar todos los autores de los que tenemos libros en **Lectums**. Si eres un fan de algún autor en particular, este es el listado que deberías consultar. Pulsa en el enlace "Autores" del menú superior, y aparecerá un listado como este:

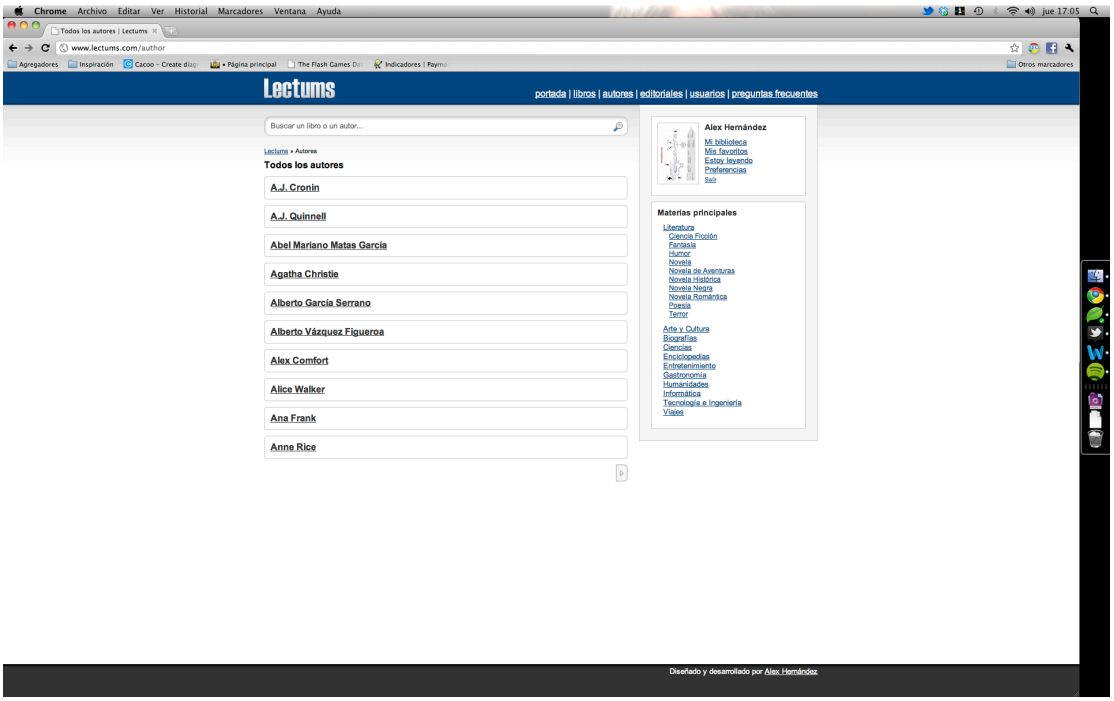

Ilustración 33 Autores

Como viene siendo habitual, si clicas en cualquier autor puedes ver su ficha personal; encontrarás todos los libros que tenemos de este autor en **Lectums**. Por ejemplo, si entráramos en la ficha de *Ken Follett*, podríamos ver la siguiente pantalla:

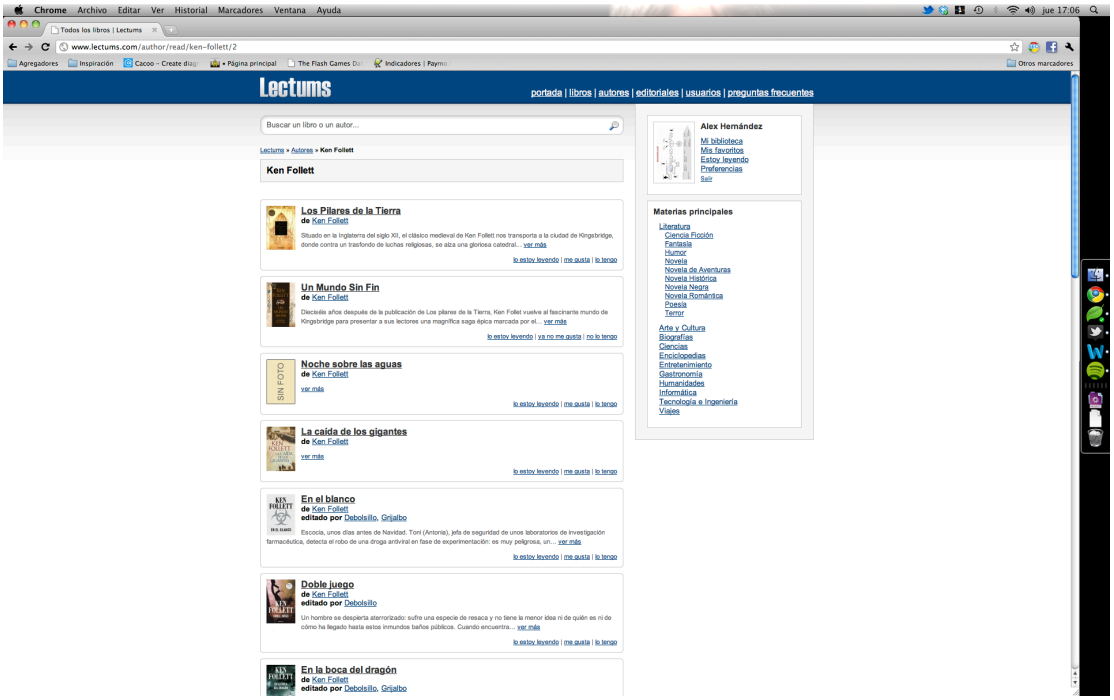

Ilustración 34 Ficha de autor

Como vemos, es un autor bastante prolífico.

8.1.1.6. Editoriales

En la sección editoriales, que podrás ver haciendo clic en el menú "Editoriales" de la parte superior, puedes ver un listado de todas las editoriales de las que tenemos libros en **Lectums**. Funciona de una forma idéntica al listado de autores, y su aspecto es el siguiente:

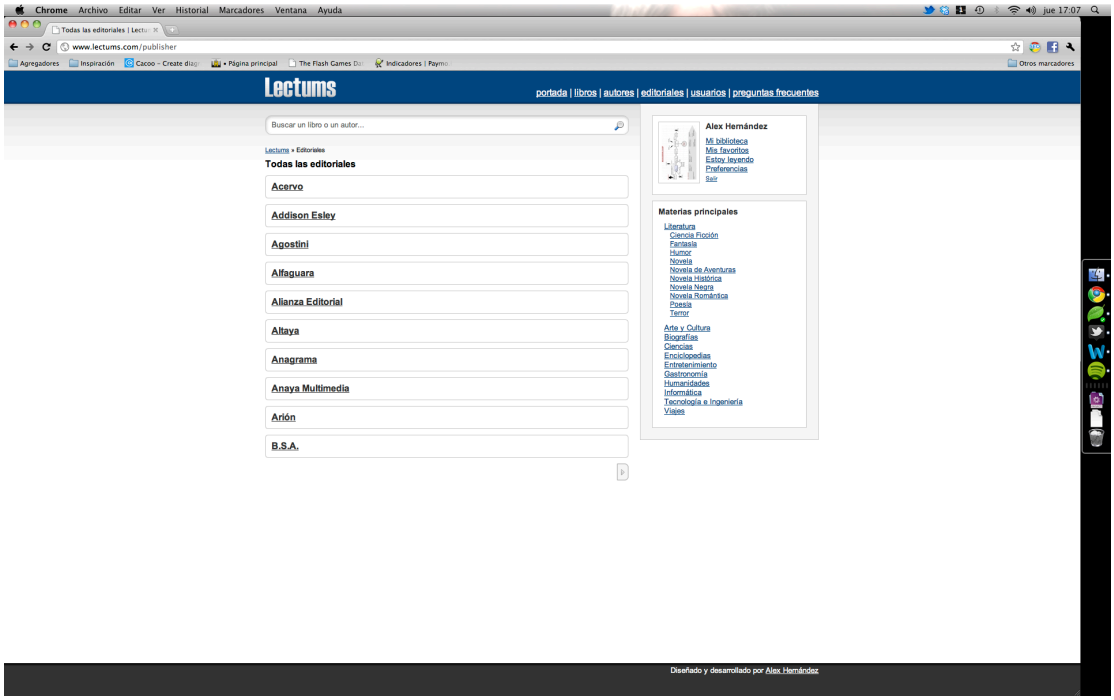

Ilustración 35 Editoriales

Si haces clic en una editorial concreta, podrás ver todos sus libros, tal como se puede ver en la siguiente imagen:

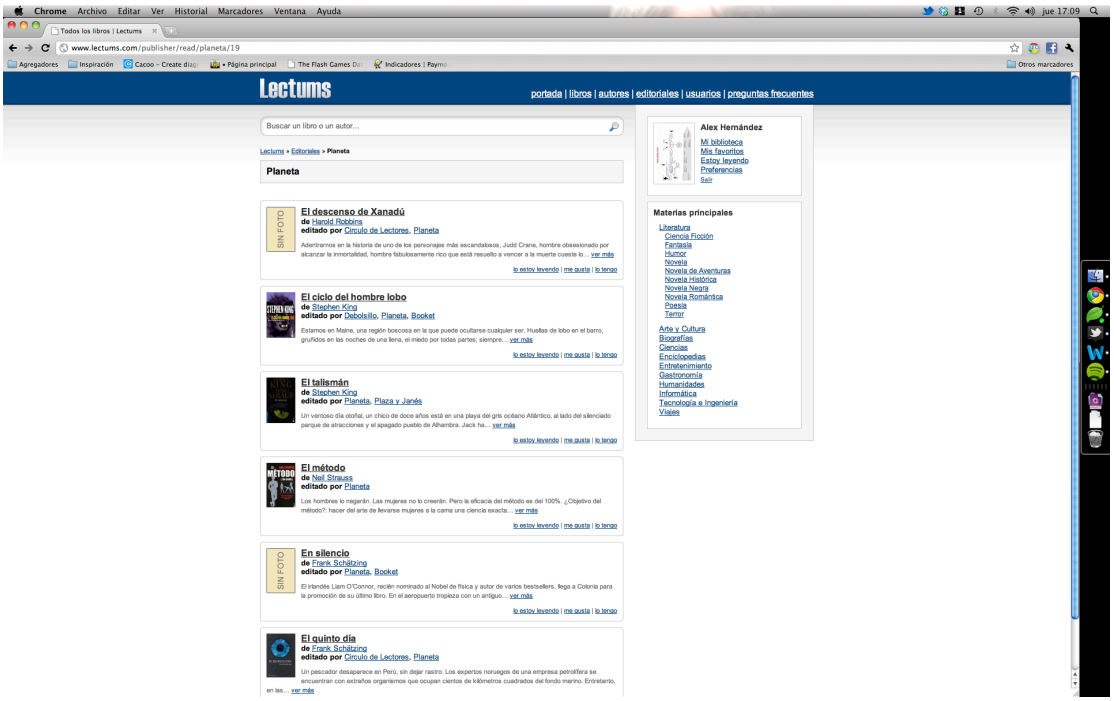

Ilustración 36 Ficha de una editorial

8.1.1.7. Usuarios

La sección de usuarios permite consultar la información de los demás usuarios de **Lectums**. Puedes ver un listado de todos ellos clicando en el enlace "Usuarios", similar a este:

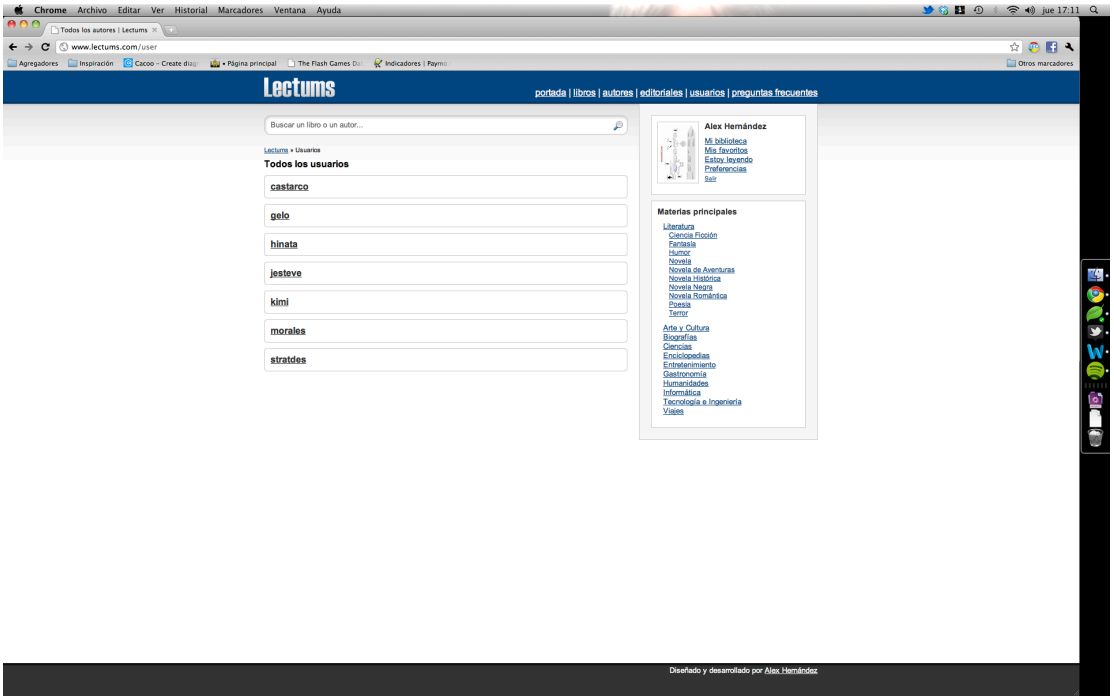

Ilustración 37 Listado de usuarios

Clicando en un usuario, por ejemplo *stratdes*, podemos ver su información; especial atención a los libros que está leyendo. Si es tu amigo, quizás podrías fiarte de sus gustos:

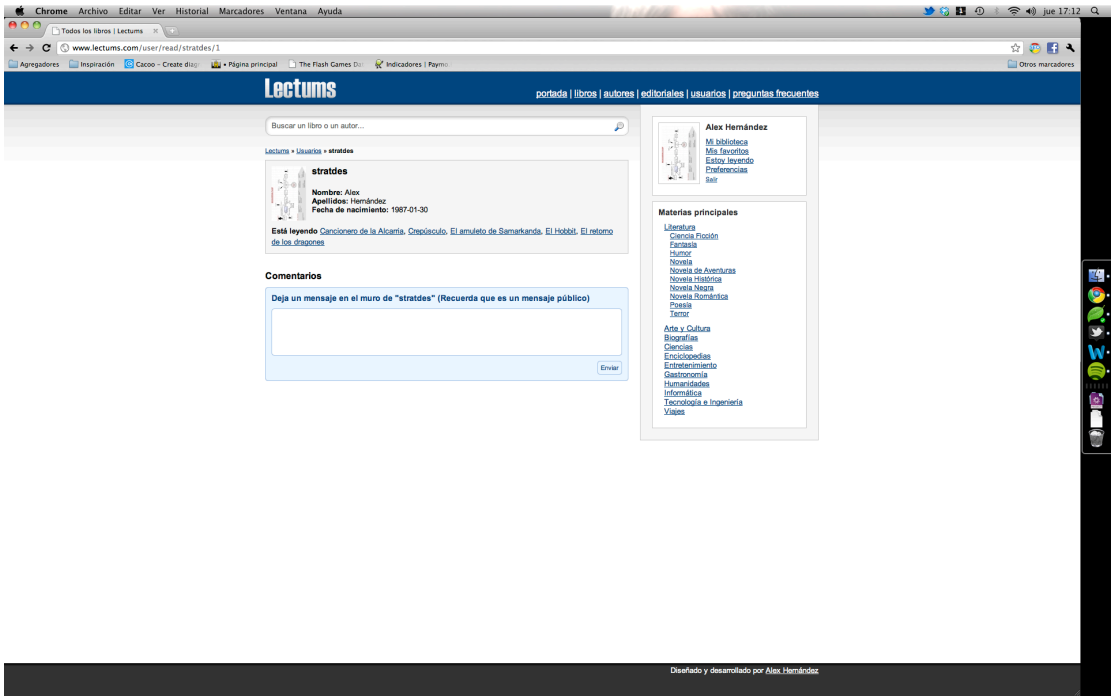

Ilustración 38 Ficha de usuario

Si aún tienes dudas, puedes escribir un mensaje en su muro preguntándole por alguno de ellos. Seguramente te podrá decir si realmente es bueno o si se está aburriendo.

8.1.1.8. Preguntas frecuentes

Las preguntas frecuentes responden algunas de las dudas más habituales de los usuarios de **Lectums**. Puedes verlas haciendo clic en "Preguntas frecuentes", en el menú superior:

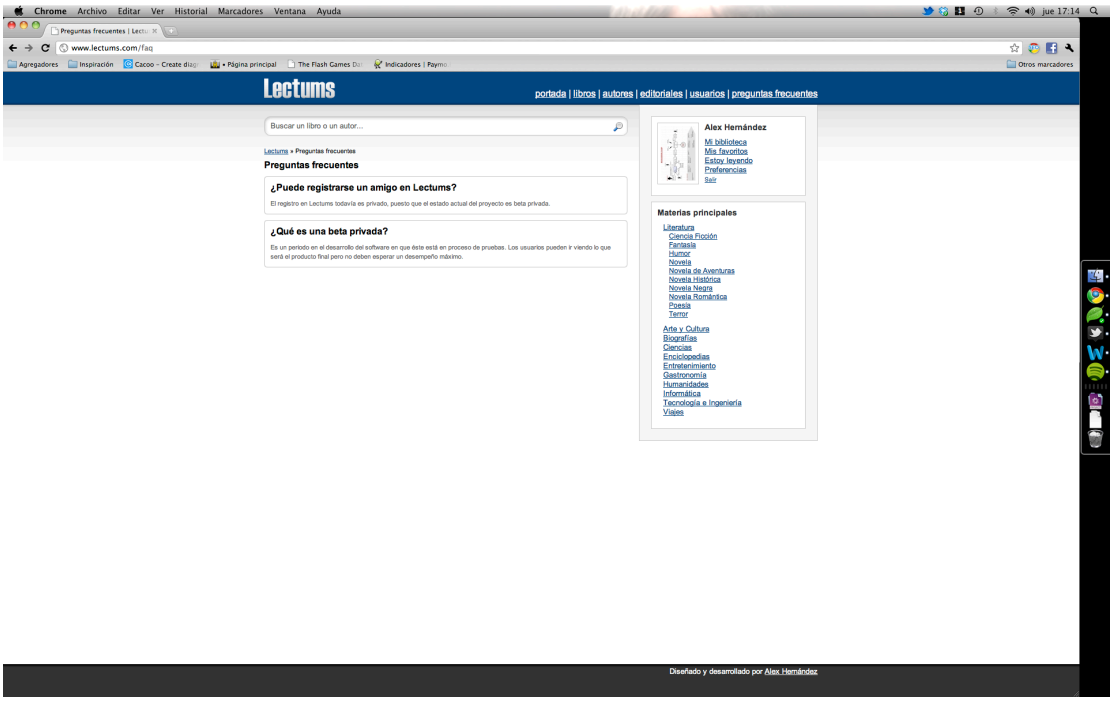

Ilustración 39 Preguntas frecuentes

8.1.1.9. Biblioteca

La biblioteca es el lugar donde puedes ver qué libros tienes, es decir, aquellos de los que hiciste clic en "lo tengo". Puedes, además, almacenar información interesante sobre la disposición de tus libros. Veamos primero una imagen de la biblioteca:

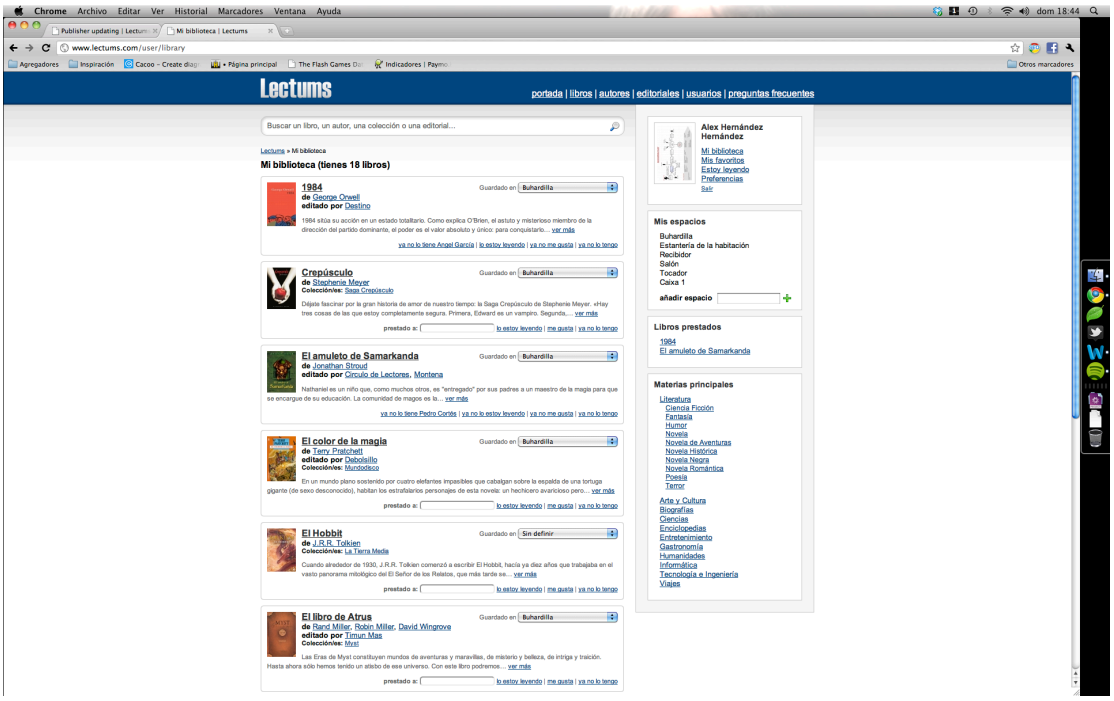

Ilustración 40 Biblioteca

Como se puede ver, en la parte superior de cada libro hay una lista desplegable: se pueden ver las ubicaciones que hemos creado para nuestro libro y definir en cual de ellos hemos guardado ese libro. ¿Cómo creamos una ubicación? Podemos ver las que hay creadas en la parte derecha de la página, bajo el título *"Mis espacios"*. Podemos escribir el nombre de un espacio nuevo y pulsar *intro* o clicar en la cruz para añadir uno. A partir de ese momento, tendremos la nueva ubicación disponible en la lista desplegable de cada libro.

Además, si en la parte derecha, debajo de *"Mis espacios"*, clicamos en una ubicación, veremos todos los libros que hemos guardado ahí.

Otra opción es especificar si has prestado algún libro a alguien; debajo de cada libro, aparece el texto *"prestado a"*, y al lado un campo de texto. Escribe el nombre de la persona y pulsa *intro*. Si escribes, por ejemplo, *Pedro Cortés* y pulsas *intro*, aparecerá el texto *"Ya no lo tiene Pedro Cortés"*. Si clicas en ese enlace, cancelarás el préstamo. Puedes ver los libros que has prestado clicando en *"Prestados"*, en la parte derecha, en la zona de usuarios.

8.1.1.10. Favoritos

En favoritos puedes ver los libros que te gustan, es decir, aquellos de los que has pulsado en el enlace "me gusta". Un buen punto en el que elegir un libro para releer si andas escaso de material nuevo:

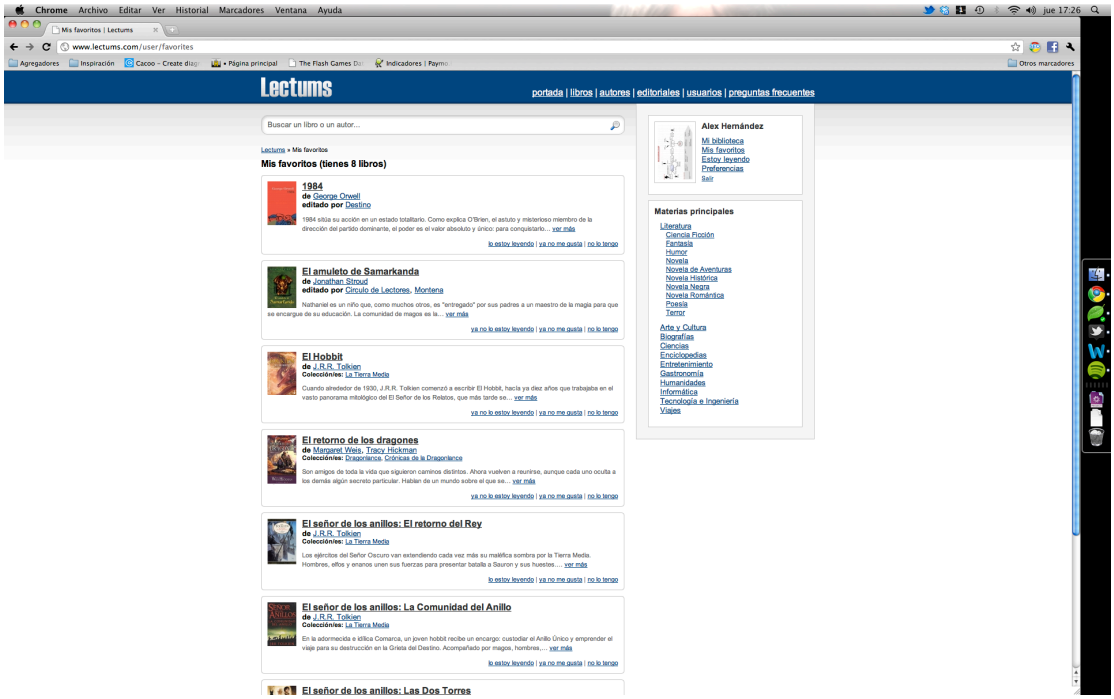

Ilustración 41 Listado de favoritos

8.1.1.11. Prestados

Puedes ver los libros prestados en la columna derecha de la sección biblioteca, tal como se ve en esta captura:

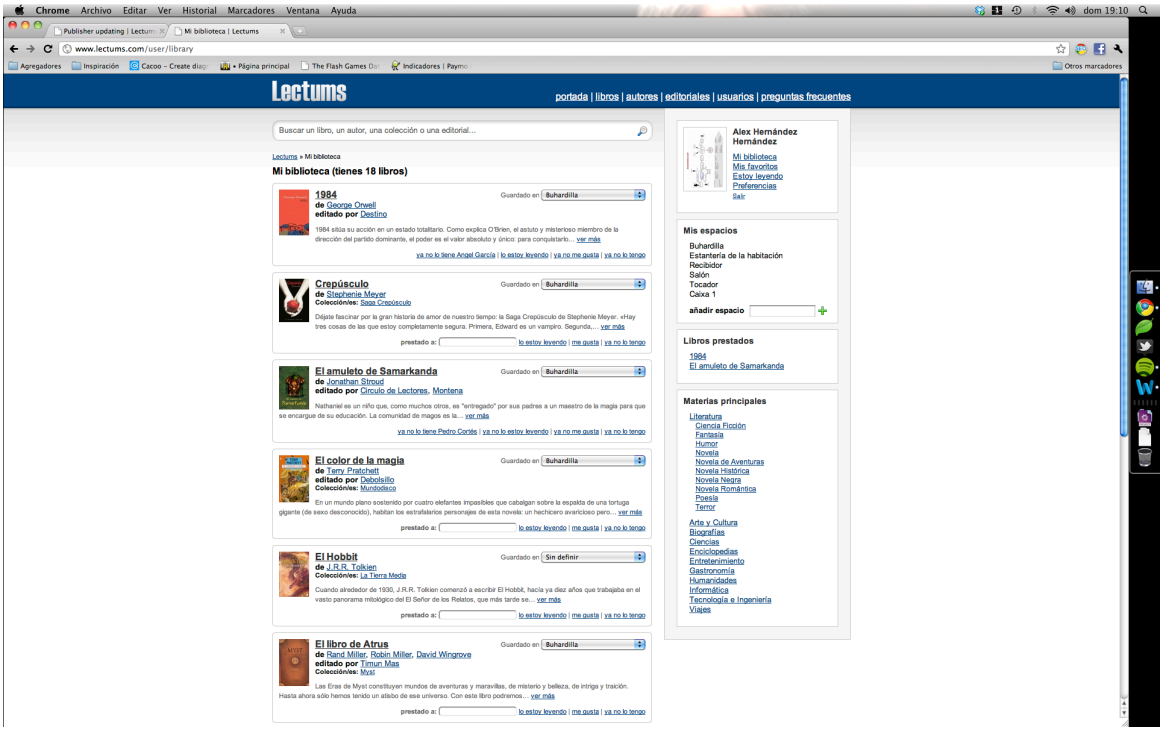

Ilustración 42 Libros prestados

8.1.1.12. Libros que estás leyendo

En libros que estás leyendo aparecen los libros que estás leyendo en este momento, es decir, aquellos de los que has pulsado el enlace “Lo estoy leyendo”. Recuerda que, además, estos libros aparecen en tu página de perfil. De esta manera, todo el mundo podrá saber lo que estás leyendo en ese momento y te podrá preguntar qué tal va la lectura:

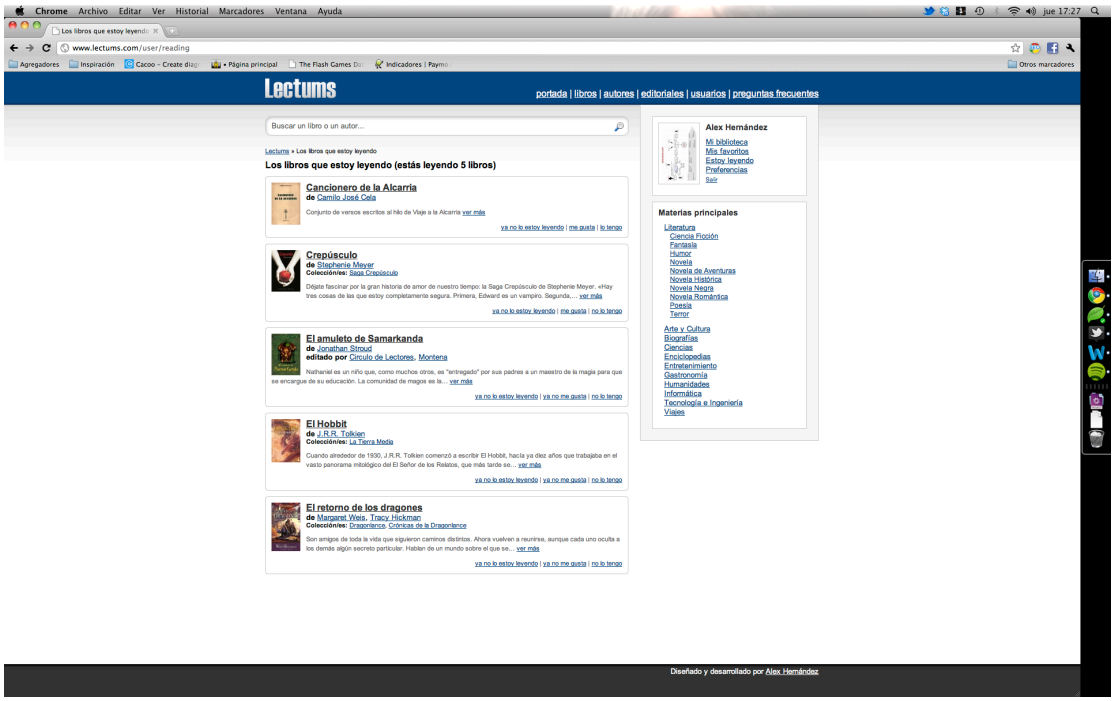

Ilustración 43 Libros que estoy leyendo

8.1.1.13. Buscador

En la parte superior de todas las páginas encontrarás un buscador. En él, puedes introducir el título de un libro, el nombre de un autor, el de una colección o el de una editorial para ver los resultados en una cómoda lista desplegable. Pulsa en alguno de los resultados y accederás directamente a la página correspondiente:

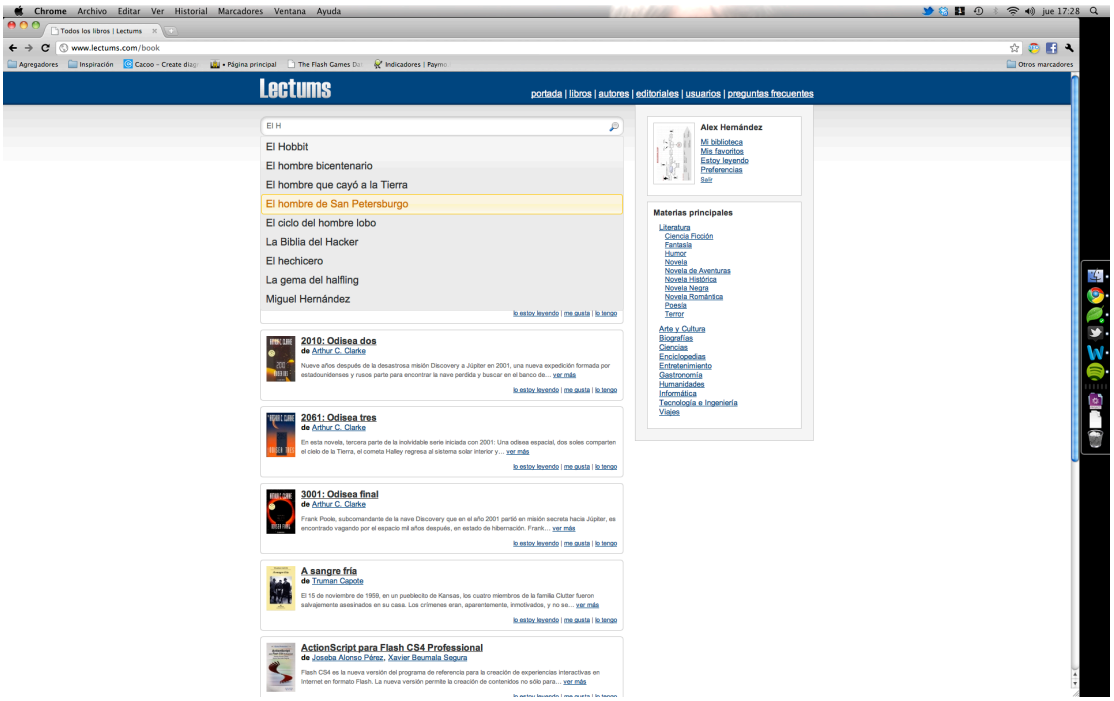

Ilustración 44 Buscador o finder

8.1.2. Manual de administrador

8.1.2.1. Instalar Lectums, la parte pública

Para instalar **Lectums** en un servidor web basta con copiar todos los archivos a la raíz y modificar los archivos *database.php*, *config.php* y añadir permisos de escritura en la carpeta "covers" y en la carpeta "avatars". En el archivo *database.php* están los datos de acceso a la base de datos. En *config.php* hay que definir la ruta base de la web. Ambos archivos están en la ruta *system/application/config*.

8.1.2.2. Instalar Lectums, el panel de administración

La instalación del panel de administración no es diferente a la de **Lectums**, sólo que en este caso no hace falta modificar permisos.

8.1.2.3. Acceder al panel de control

Para acceder al panel de administración, hay que acceder a la url *www.lectums.com/admin*, en el momento de escribir esta memoria. Nos aparecerá la siguiente pantalla, que nos solicita usuario y contraseña:

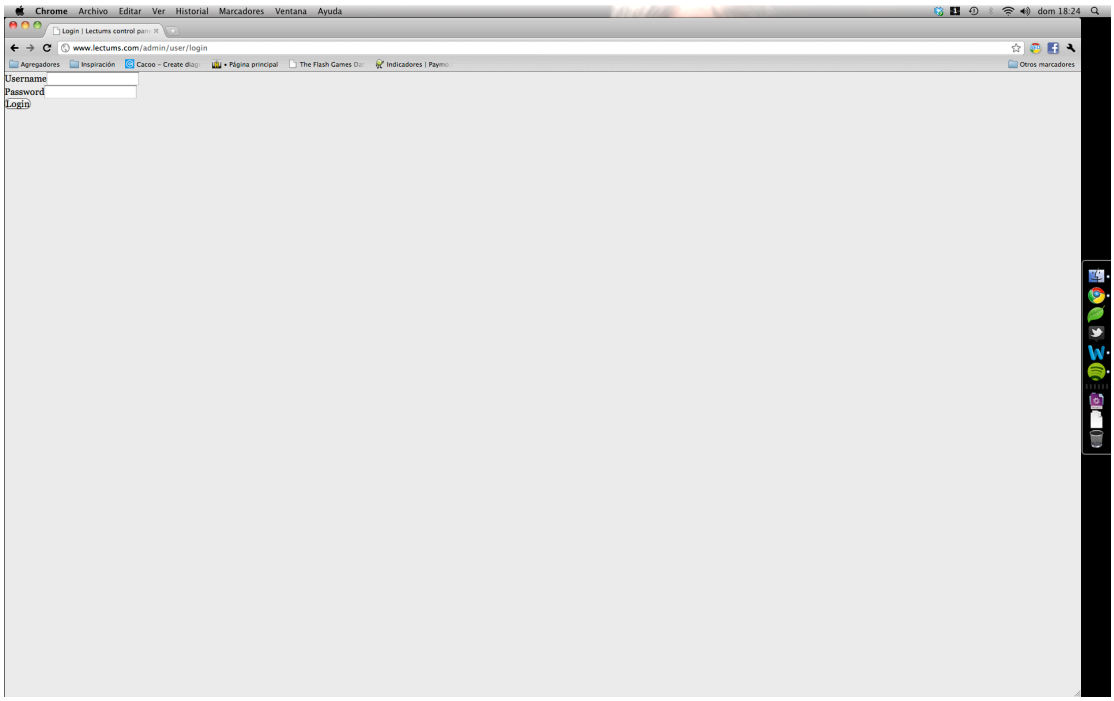

Ilustración 45 Pantalla de Login del administrador

Tras introducir los datos correctos, nos redirigirá al listado de libros, que es el utilizado más frecuentemente:

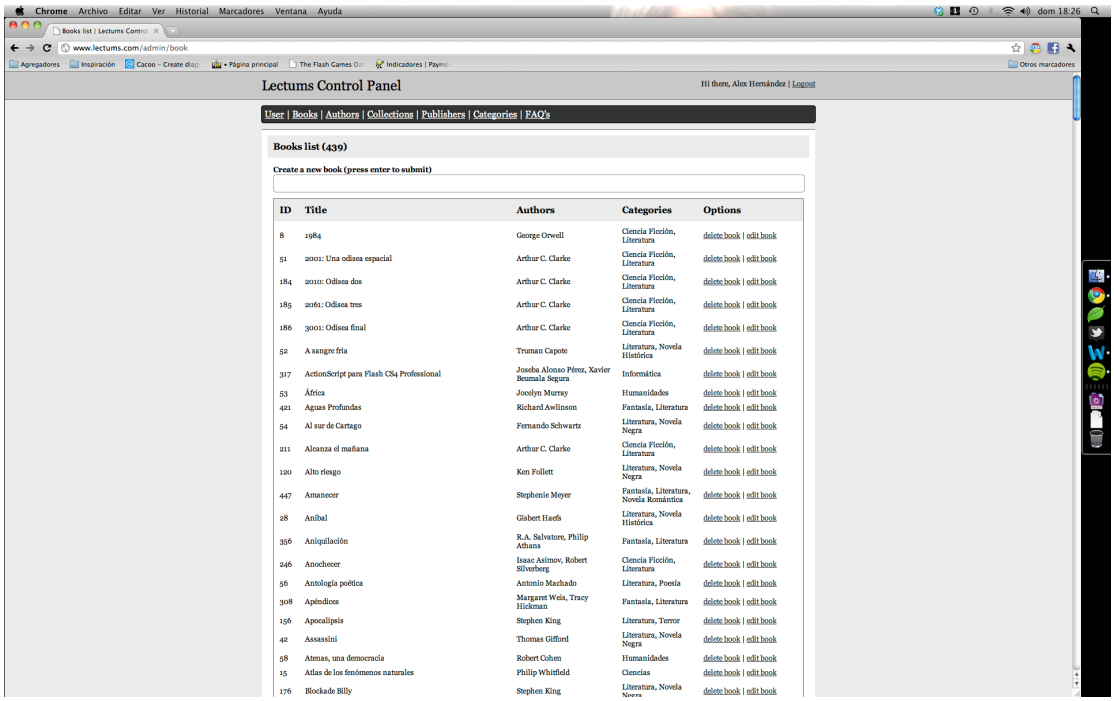

Ilustración 46 Listado de libros

8.1.2.4. Crear un libro

Para crear un libro, debemos introducir el título en el cuadro de texto de la parte superior, justo debajo de la etiqueta *create a new book(press enter to submit)*. Tras introducir el título, pulsamos la tecla *enter*. Nos aparecerá nuestro nuevo libro, sin más datos, en el listado inferior.

8.1.2.5. Modificar un libro

Para modificar un libro debemos pulsar en su botón *edit* correspondiente (esto es, el que está en la misma línea). Si, por ejemplo, modificamos el libro *1984*, veremos la siguiente pantalla:

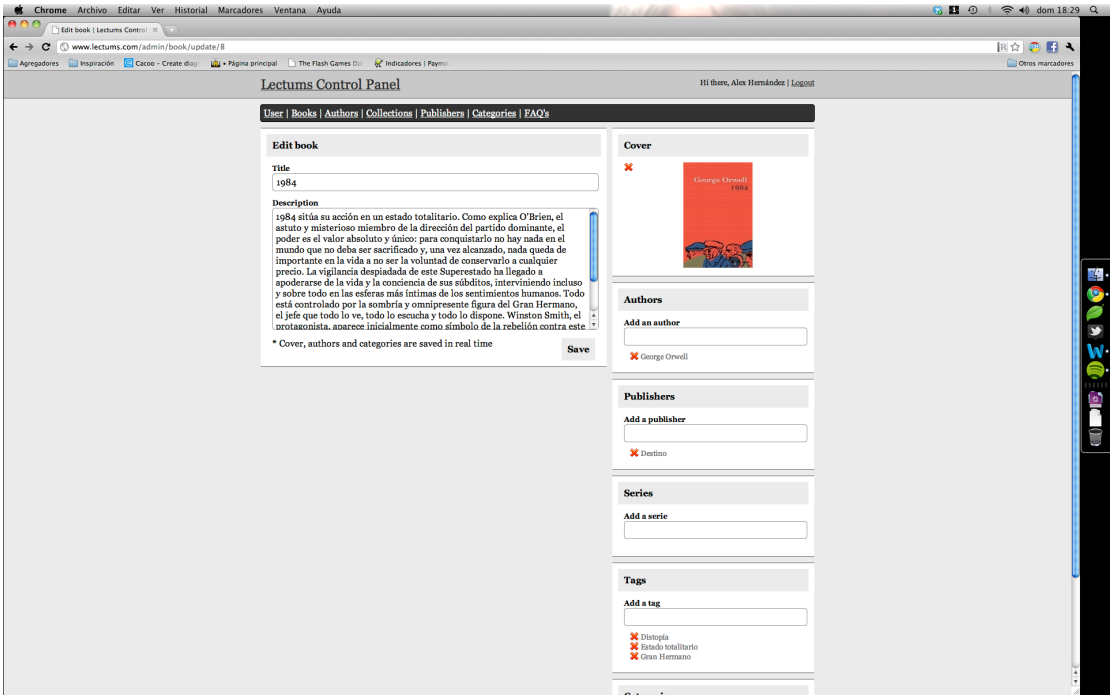

Ilustración 47 Modificando un libro

En esta pantalla hay dos campos fundamentales, título y descripción. Para modificarlos, basta cambiar el texto y pulsar la tecla *save*. Una vez hecho, volvemos al listado de libros. Si volvemos a pulsar en *edit*, veremos que los datos se han modificado correctamente.

Para modificar la portada, primero la borramos pulsando en la cruz roja. Nos aparecerá un cuadro de selección de fichero: seleccionamos el fichero correcto, pulsamos aceptar y la imagen se introducirá. No es necesario pulsar el botón *save* esta vez, es automático y funciona mediante AJAX.

Podemos eliminar los autores pulsando en sus cruces rojas; para añadir uno nuevo, hay que escribir su nombre en el cuadro de texto situado justo debajo de *add an author*: en caso de existir ya, nos aparecerá en un listado, tal como se ve en la figura inferior. En ese caso, hemos de clicar

en la lista. Si no existe, no aparecerá, y simplemente debemos pulsar *intro*. A partir de ese momento, aparecerá en los listados. Este es el primer método de introducción de autores.

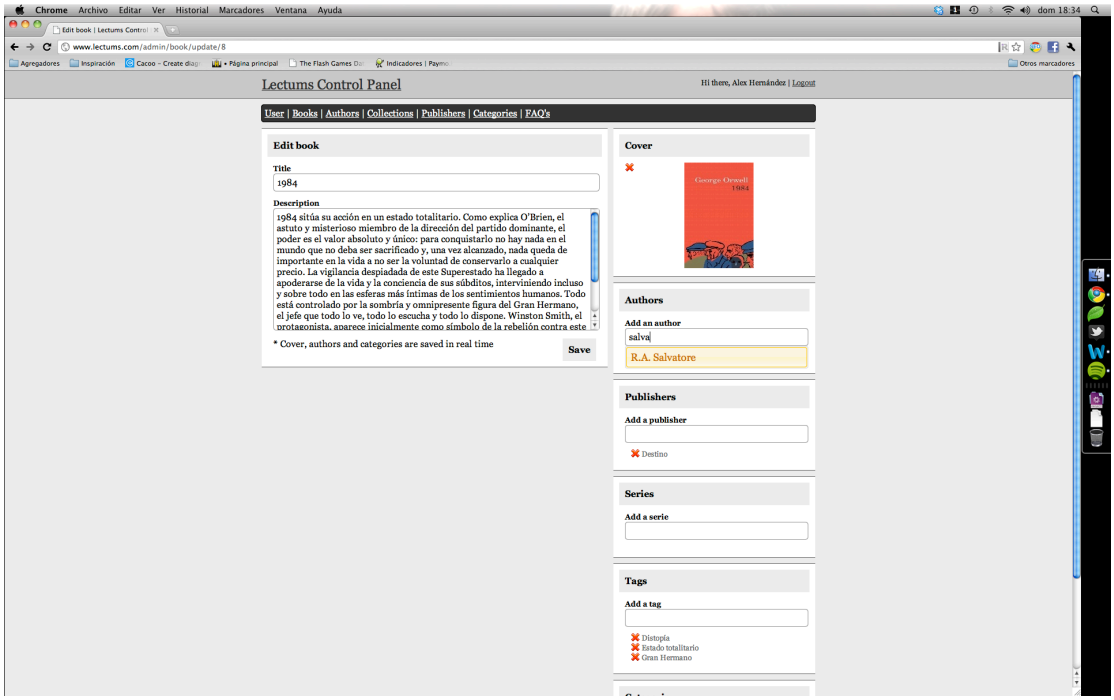

Ilustración 48 Añadiendo un autor

Podemos seguir el mismo método para añadir editoriales, colecciones (series), etiquetas (aunque en este prototipo no aparecerán en la parte pública) y categorías (no es recomendable añadir categorías nuevas, deberían bastar las existentes.

8.1.2.6. Borrar un libro

Borrar un libro es tan sencillo como pulsar en enlace *delete* correspondiente. Volveremos al listado y habremos eliminado el libro.

8.1.2.7. Crear otros elementos

Podemos crear autores, editoriales, categorías, FAQ's, etc, de una forma muy sencilla. Si por ejemplo queremos crear una nueva editorial, pulsamos el enlace *publishers* del menú superior. Aparecerá el listado de editoriales, tal como se ve en la siguiente pantalla:

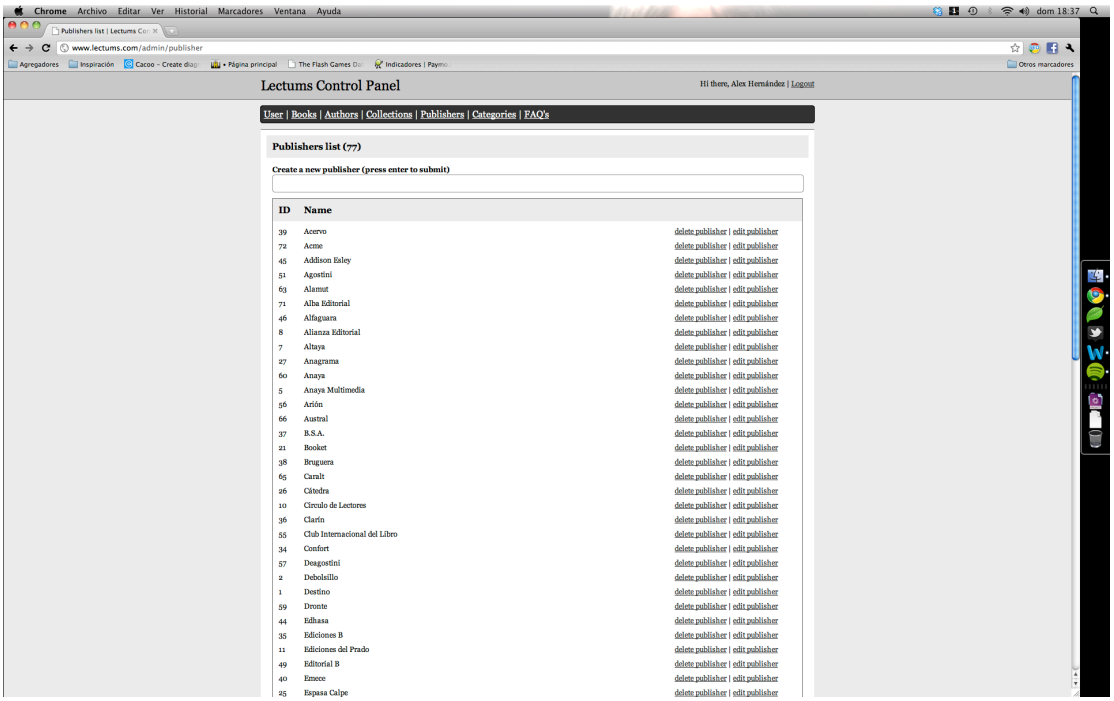

Ilustración 49 Listado de editoriales

Escribimos el nombre de la editorial en el campo de texto y pulsamos *intro*. Como no hay más datos que añadir a la editorial, ya hemos completado todo el proceso y veremos la nueva editorial en el listado inferior. Este proceso es idéntico para el resto de elementos. Lo que introduzcamos de esta manera, también aparecerá cuando modifiquemos libros, en el correspondiente listado.

8.1.2.8. Modificar otros elementos

Para modificar un elemento, por ejemplo la editorial que añadimos antes, debemos pulsar en su enlace *edit* correspondiente. Vemos la siguiente pantalla.

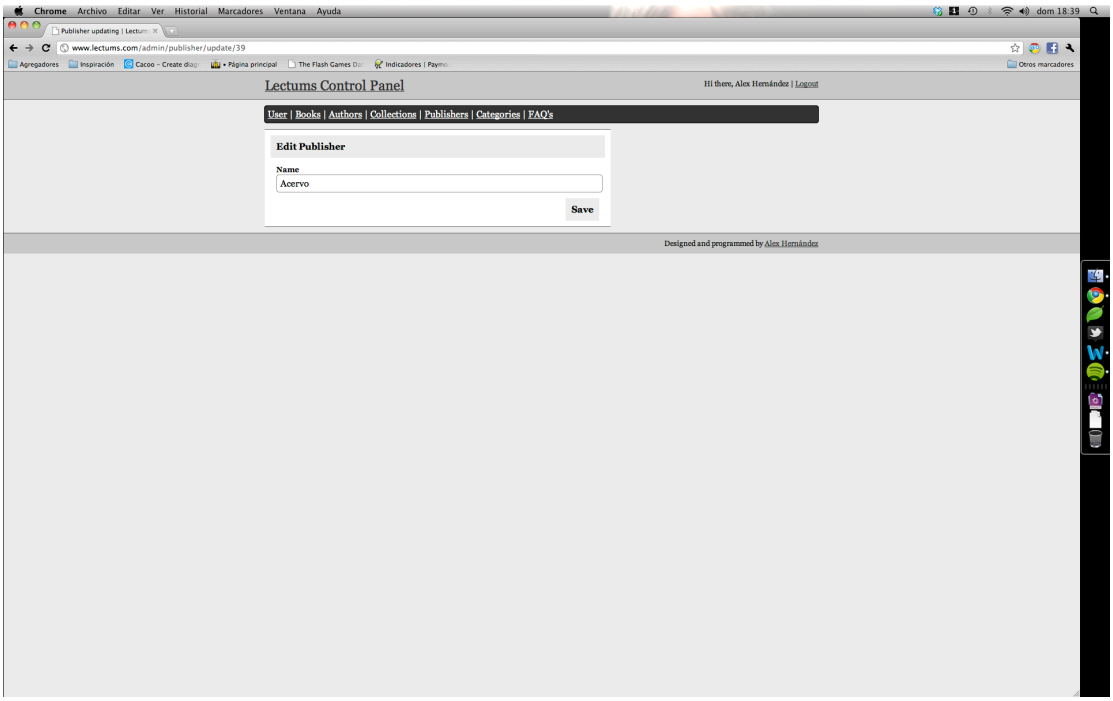

Ilustración 50 Modificando una editorial

Cambiamos el nombre y pulsamos *save*. Listo.

8.1.2.9. Borrar otros elementos

Para borrar cualquier elemento, pulsamos en el botón delete correspondiente. ¡Esta operación no se puede deshacer!

8.2. Apéndice B: Codeigniter, estructura y metodología

8.2.1. ¿Qué es Codeigniter?

Codeigniter es un framework de desarrollo web basado en el paradigma MVC y escrito íntegramente en PHP. Los requisitos para poder desarrollar una aplicación en Codeigniter son PHP 4.3.2 o superior; en el uso habitual de Codeigniter requeriremos una base de datos de entre una de las siguientes: MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite, and ODBC.[8]

Entre sus características principales encontramos las siguientes:

- Es libre y gratuito, y su licencia es Apache/BSD
- Es ligero: esto significa que su rendimiento es ligeramente superior al resto de frameworks PHP, aunque a veces sea a costa de renunciar a ciertas funcionalidades.

8 Requisitos de Codeigniter http://codeigniter.com/user_guide/general/requirements.html

- Generar URL's amigables: siguiendo la línea de todos los frameworks MVC, Codeigniter genera URL's cargadas de semántica, especialmente interesantes de cara al posicionamiento.
- Es extensible: es realmente sencillo extender el núcleo de Codeigniter según nuestras propias necesidades.
- Codeigniter está bien documentado, sobre todo en comparación con otras aplicaciones de Software libre.
- Tiene una comunidad detrás bastante importante que apoya su crecimiento.

8.2.2. ¿De dónde proviene?

Codeigniter es un producto de Ellislab, y es la base del gestor de contenido –de pago– ExpressionEngine. En principio ambos productos se benefician mutuamente en su desarrollo, aunque obviamente Ellislab potencia su producto de pago por encima de su producto gratuito.

8.2.3. ¿Quién utiliza Codeigniter?

Desde pequeños desarrolladores hasta grandes compañías, pasando por aplicaciones de software libre tan interesantes como MyClientBase o PyroCMS. Es uno de los frameworks en PHP más utilizados, y su tendencia en cuanto a popularidad ha aumentado linealmente, como se puede comprobar fácilmente en Google Trends.

8.2.4. ¿Y es difícil aprender Codeigniter?

Pues realmente no. La curva de aprendizaje de este framework de desarrollo web es bastante asumible, siendo más baja que la de la mayoría de frameworks en PHP; es especialmente notoria la diferencia con CakePHP, siendo este último bastante más difícil de aprender. No obstante, y como en cualquier otro ámbito de la vida, requiere cierto esfuerzo llegar a dominar plenamente el framework.

8.2.5. ¿Codeigniter va a seguir creciendo?

Todo parece indicar que sí, y en Ellislab ultiman la salida de la versión 2.0 de Codeigniter, que lleva un tiempo parada en la 1.7.3, aunque el código de la versión 2.0 lleva en beta ya un tiempo. ¿Qué quiere decir esto? Que durante unos pocos meses ha habido dos ramas bastante estables, y que en Ellislab han decidido impulsar definitivamente la 2.0 y a partir de su lanzamiento oficial volverá el ciclo habitual de lanzamientos. Por tanto, y a no ser que ocurra algo extraño –y sería absurdo a tenor del éxito de este framework- Codeigniter va a seguir siendo uno de los mejores frameworks de desarrollo existentes.

8.2.6. Estructura o Workflow de Codeigniter

El siguiente esquema, de la documentación de Codeigniter, ilustra perfectamente el flujo de trabajo –desde que se accede a una URL hasta que se muestra en pantalla- que sigue Codeigniter:

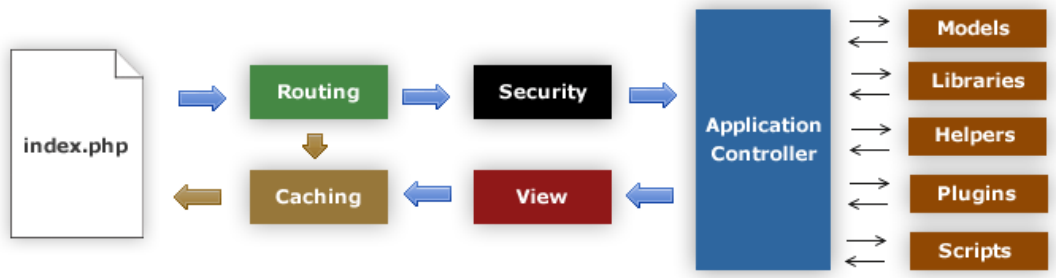

Ilustración 51 Flujo de Codeigniter

Vemos primero de todo que cualquier petición, cualquier URL, accede al archivo index.php. Da lo mismo cual sea, da lo mismo si parece que accedemos a otra carpeta; la realidad es que accedemos a index.php. ¿Y qué ocurre entonces? En función de nuestra URL, se produce un routing, esto es, se selecciona el archivo correcto a cargar; pero no siempre así: en el caso de existir una cache válida, entonces se carga y no se procesa absolutamente nada.

De no haber cache, se selecciona el controlador y el método adecuados: en este punto, existen muchas posibilidades, como se ven en los cuadros marrones. Habitualmente, se hace alguna consulta a algún modelo, que devuelve datos de la base de datos ya procesados. En ese momento, se carga la vista correspondiente, a la que se pasan esos datos.

Por último, la vista se muestra en pantalla como corresponda y el navegador recupera el control; si procede copiar esa vista en la cache para otros accesos, este es el punto donde se hace -postproceso, es decir, después de procesar datos.

8.2.7. Metodología habitual

Vista la estructura y el workflow de Codeigniter, podemos asumir que la metología habitual consiste en que cualquier petición que queramos hacer debe pasar obligatoriamente por un controlador que, a su vez, solicitaré todos los datos necesarios al modelo; por último, la vista mostrará esos datos correctamente al usuario.

Por lo tanto, es fácil suponer que cualquier caso de uso involucra, necesariamente, a un controlador y a una vista; en la mayoría de los casos, siempre que haya acceso a la capa de datos, involucrará también a un modelo. Visto de otra manera, cada caso de uso tiene al menos un método de un controlador y una vista asociados.

8.2.8. Librerías y Helpers

Codeigniter, como todos los frameworks de desarrollo web, dispone de una serie de librerías y helpers o ayudantes que facilitan algunas tareas rutinarias, como el control de sesiones, el acceso a base de datos, etc. Quiero destacar algunas de ellas que son imprescindibles para el correcto funcionamiento de Lectums.

- *Database*: es la librería que permite la conexión a la base de datos. Tiene una buena dosis de funciones automáticas para el acceso y la gestión de datos, así como herramientas de seguridad que impiden –relativamente- los ataques de SQL injection.

- *Session*: es la librería que permite crear y gestionar sesiones; una sesión no es más que dotar de persistencia a un medio –la web- que de por sí no es persistente, al carecer de estado. En la sesiones se guardan variables con información tal como el nombre de usuario, la id, si ha hecho login, cuándo…

- *URL:* permite trabajar con la URL, entre otras cosas obtener distintas partes de esta; importante, por ejemplo, para saber cuál es el controlador, los atributos… Además, permite saber la ruta base del proyecto –muy importante para generar links; de hecho, permite generarlos automáticamente con sólo facilitar el controlador, el método y los atributos.

8.3. Apéndice C: Software utilizado

Para la realización de este proyecto se han utilizado una serie de aplicaciones que vale la pena destacar. Ya hablé en la planificación temporal y económica de Things para la gestión de tareas. Además de ello, Lectums se ha desarrollado sobre MacOS, aunque el servidor web en el que está alojado está basado en Linux. Coda, Textmate, MAMP… son sólo otras de las aplicaciones utilizadas. Hablemos con más detalle de ellas:

8.3.1. MacOS

MacOS ha sido el sistema operativo escogido para todo el desarrollo. Basado en Unix, hay poco que se pueda decir que no se sepa ya de este brillante sistema operativo. En particular, me ha dotado de una productividad muy superior a la que me hubiera proporcionado Windows o Linux; es una cuestión personal, obviamente, pero es una de las decisiones que más he agradecido.

Ilustración 52 MacOS

8.3.2. Things

Things es un software gtd –getting things done- que me ha permitido gestionar mis tareas de una forma muy sencilla. Como decía en la sección de tiempos, Things es una aplicación de pago, pero el desembolso merece la pena. Su flexibilidad y facilidad de uso aumentan la productividad de aquellas personas que requieren un control de sus tareas.

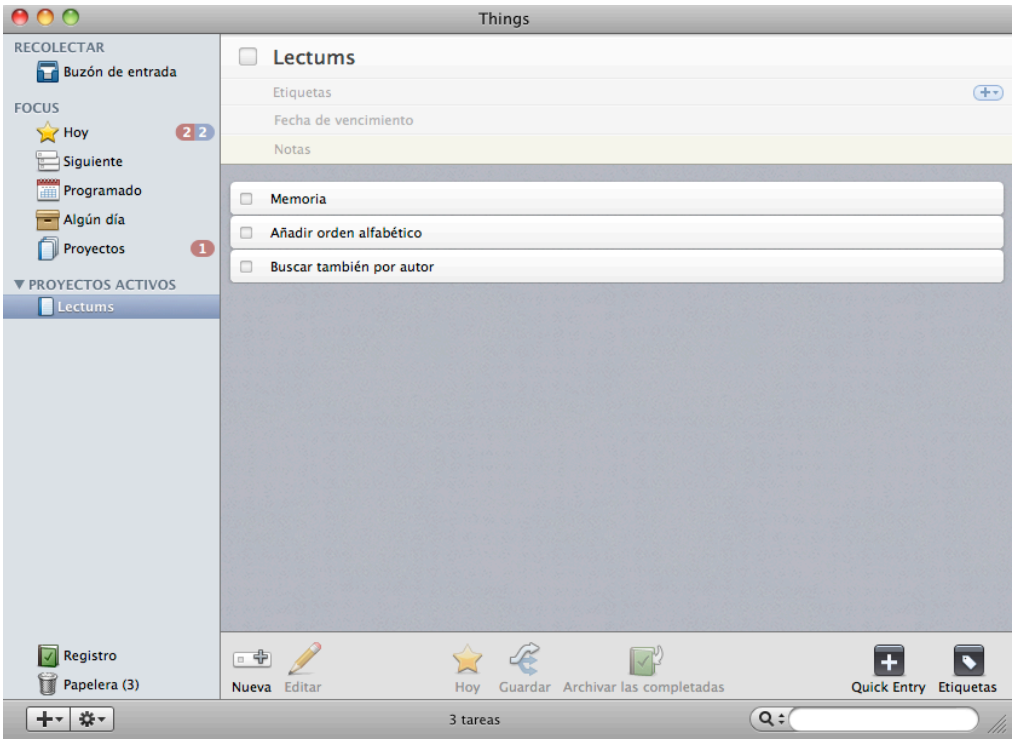

Ilustración 53 Things

8.3.3. Coda

Coda es una suite de desarrollo web de gran calidad similar a Dreamweaver que sólo está para Mac. Entre otras muchas cosas, permite una sincronización total con el servidor, aparte del uso de snippets, subversión…

Tampoco es gratuita, una licencia vale $99.

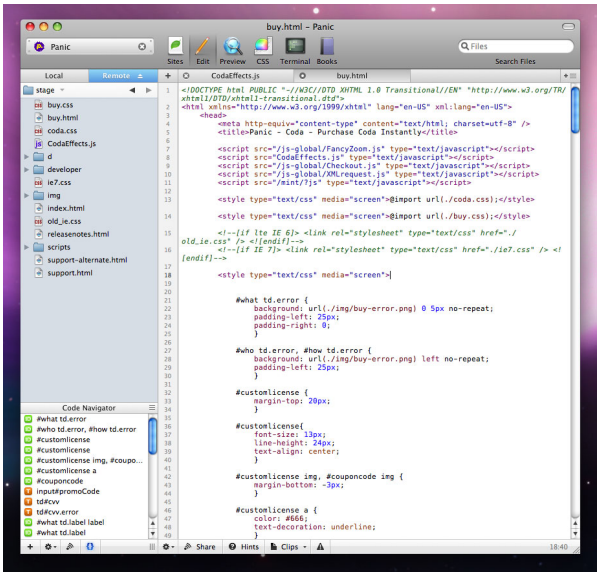

Ilustración 54 Coda

8.3.4. Transmit

Transmit es una aplicación FTP para Mac de gran calidad. Permite incluso crear unidades virtuales asociadas a un servidor FTP. Es mi cliente por defecto. ¿Su precio? $34

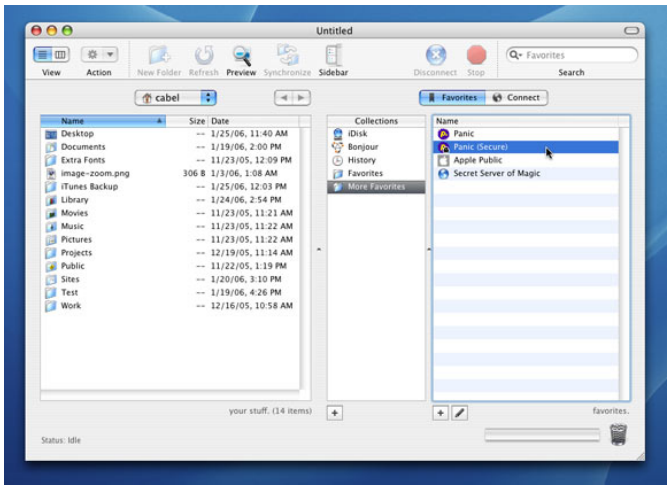

Ilustración 55 Transmit

8.3.5. MAMP

MAMP son las siglas de Mac, Apache, MySQL y PHP, y es un servidor todo en uno para hacer pruebas en local. Pese a que el servidor web es ostensiblemente más rápido que una máquina standard haciendo de servidor, un servidor local puede resultar muy útil en determinadas circunstancias. MAMP tiene una versión gratuita, que es la que utilizo yo, y una de pago, con algunas funcionalidades extra.

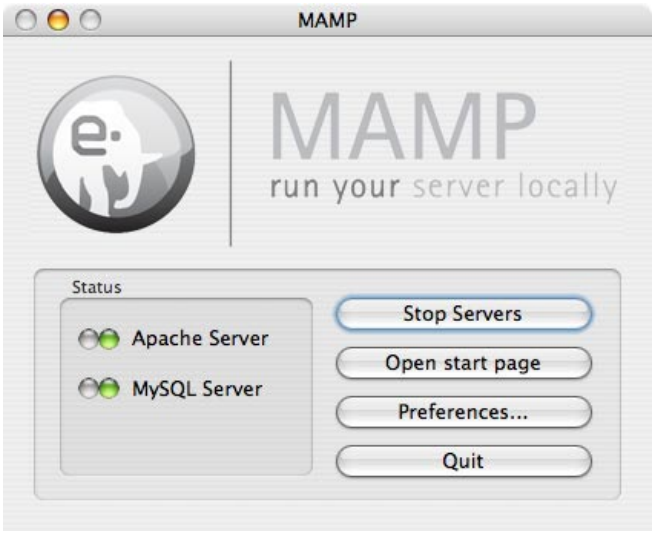

Ilustración 56 Mamp

8.3.6. Photoshop

Photoshop es un software de diseño gráfico ampliamente utilizado por los profesionales del diseño, tanto del diseño web como del diseño gráfico en general; hay versión tanto para Windows como para Mac.

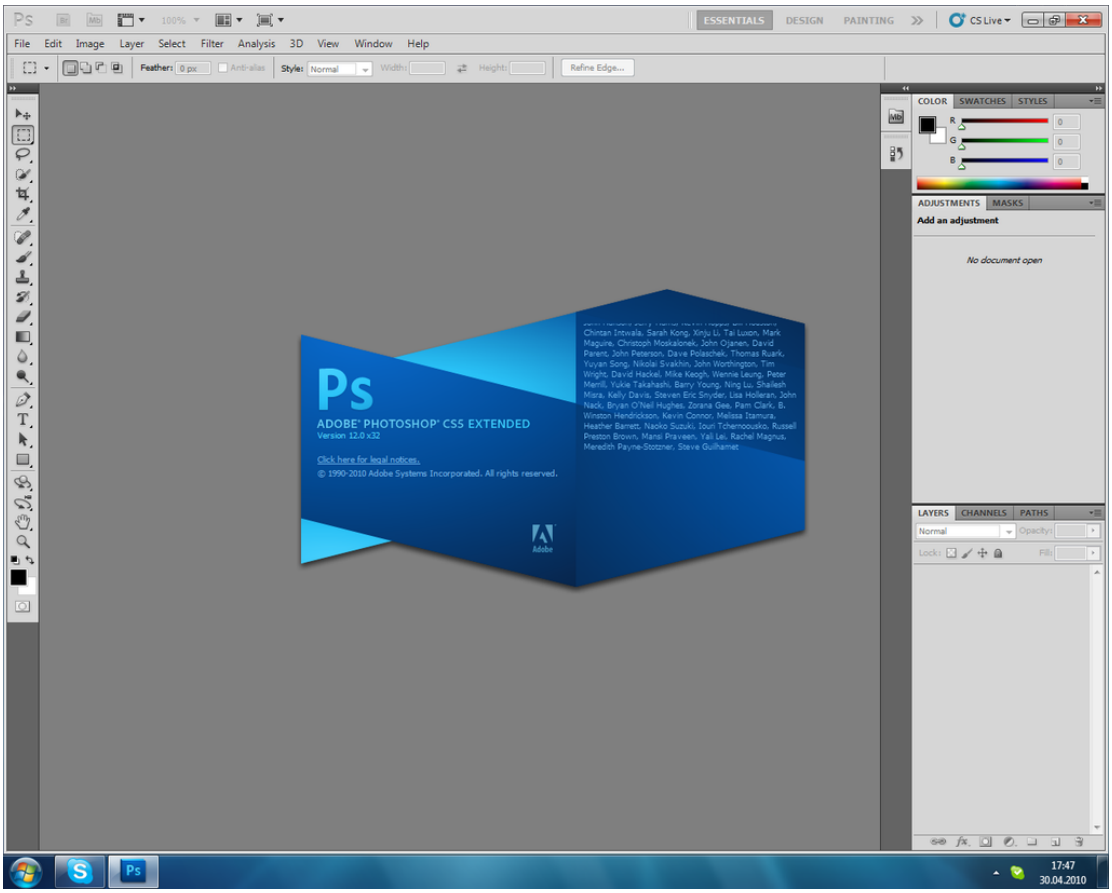

Ilustración 57 Photoshop

8.4. Apéndice D: Hosting web

Para llevar a cabo Lectums no sólo han sido necesarias una serie de herramientas de software sino también un servidor web que me permitiera alojar el proyecto durante las pruebas y, en caso de lanzarlo, durante el funcionamiento de este. La empresa que aloja actualmente Lectums es Dreamhost; no obstante, la decisión no ha sido trivial, y se basa en una serie de requisitos funcionales y de servicio:

- Un servidor web que acepte PHP y MySQL
- Un servidor web con una capacidad no menor a 2GB
- Un servidor web con una transferencia capaz de soportar unas 20.000 visitas mensuales y unas 800.000 páginas vistas (cálculos orientativos teniendo en cuenta cierto éxito).

Hice una comparativa de los diferentes servicios que conocía y por precio y funcionalidades escogí Dreamhost; recojo a continuación, no obstante, dicha comparativa a modo informativo:

OVH

- Precio: 23,18€/mes IVA incl.
- Espacio: 500GB
- Transferencia: 5TB
- Velocidad del servidor: rápida, lo he probado.
- Bases de datos: 4, 3 con 300MB y 1 con 1GB.

Dreamhost

- **Precio: 9€/mes IVA incl.**
- **Espacio: Ilimitado**
- **Transferencia: Ilimitado**
- **Velocidad del servidor: Buena, lo he probado.**
- **Bases de datos: Ilimitadas con espacio ilimitado.**

1and1

- Precio: 23,18€/mes IVA incl.
- Espacio: 20GB
- Transferencia: Ilimitado
- Velocidad del servidor: desconocida.
- Bases de datos: 50 de 100MB.

CDMon

- Precio: 23,18€/mes IVA incl.
- Espacio: 7.5GB
- Transferencia: 200GB
- Velocidad del servidor: desconocida.
- Bases de datos: Ilimitadas, 500MB.

Arsys

- Precio: 23,18€/mes IVA incl.
- Espacio: 500GB
- Transferencia: 5TB
- Velocidad del servidor: rápida, lo he probado.
- Bases de datos: 4, 3 con 300MB y 1 con 1GB.

Hostalia

- Precio: 22.90€/mes IVA incl.
- Espacio: 2GB
- Transferencia: 40GB
- Velocidad del servidor: rápida, lo he probado.
- Bases de datos: 20, tamaño desconocido.

9. Glosario

A

ASP, 15, 115

C

Caso de uso, 23, 30, 32, 33, 34, 45, 46, 104
Codeigniter, 5, 16, 17, 23, 39, 40, 76, 77, 102, 103, 104, 105
CSS, 5, 21, 22, 23, 77, 115

F

Framework, 15, 16, 17, 20, 21, 22, 23, 39, 40, 41, 45, 66, 102, 103

G

Google, 15, 49, 69, 71, 72, 73, 74, 75, 76, 78, 80, 103

H

HTML, 21, 45, 56
HTML5, 21, 22

J

JavaScript, 21, 22, 48, 50, 115
JQuery, 5, 21, 22, 50, 51

L

LAMP, 14, 15, 16

M

Modelo conceptual, 19, 28, 29
Modelo de Componentes, 39, 40
MVC, 16, 17, 18, 19, 23, 39, 52, 76, 79, 102, 103

O

ORM, 16

P

PHP, 5, 14, 15, 16, 63, 102, 103, 108, 109, 115

R

Ruby on Rails, 15, 115

W

Web 2.0, 5, 65, 66

X

XHTML, 5, 21, 22, 23, 67, 73, 77

10. Bibliografía

Todo proyecto requiere de unas fuentes que permitan al autor documentarse con el objetivo de poder desarrollar el tema del que se habla. Además, estas fuentes sirven de apoyo para su argumentación. Para hacer esta memoria he utilizado algunos libros y enlaces de Internet, que expongo a continuación separándolos por tema:

10.1. Desarrollo de la lógica

- La biblia de PHP 5, de John Coggeshall, Anaya Multimedia, 9788441518452
- Proyectos profesionales, PHP, de varios autores, Anaya Multimedia, 9788441514188
- JavaScript, de Jim Keogh, Anaya Multimedia, 9788441519596
- Ruby on Rails, de Bruce A. Tate y Curt Hibbs, Anaya Multimedia, 9788441521827
- ASP.NET 4.0 (Saltando desde la versión 2.0), de José Manuel Alarcón Aguín, Krasis Press, 9788493669614
- Información sobre PHP en desarrolloweb.com (http://www.desarrolloweb.com/php/)

10.2. Diseño web

- Diseño de páginas web, Jack Davis y Susan Merritt, Anaya Multimedia, 9788441509498
- Inspiración CSS remix (http://www.cssremix.com)

10.3. Maquetación

- Principios de diseño web, de Jeffrey Zeldman, Anaya Multimedia, 9788441513440
- W3C Schools (http://www.w3schools.com/)

10.4. Usabilidad

- Información sobre usabilidad (varias guías temáticas) en desarrolloweb.com (http://www.desarrolloweb.com/directorio/diseno/usabilidad/).

10.5. Accesibilidad

- Accesibilidad por el W3C (http://www.w3c.es/divulgacion/guiasbreves/accesibilidad) y otros del propio W3C

10.6. Productividad y negocios

- Rework, change the way you work forever, de Jason Fried y David Heinemeier Hansson, Vermillion, 9780091929787

- Getting real, de Jason Fried, David Heinemeir Hansson y Matthew Linderman, ebook gratuito (http://gettingreal.37signals.com/toc.php)

- Loogic blog (http://www.loogic.com)

10.7. General

- A list apart blog (http://www.alistapart.com)